COOPERATIVE DATABASE SYSTEMS

Consider posing a query to a human expert. If the posed query has no answer or the complete data for an answer are not available, one does not simply get a null response. The human expert attempts to understand the gist of the query, to suggest or answer related questions, to infer an answer from data that are accessible, or to give an approximate answer. The goal of cooperative database research is to create information systems with these characteristics (1). Thus, the system will provide answers that cooperate with the user. The key component in cooperative query answering is the integration of a knowledge base (represents data semantics) with the database. Research in cooperative answering stems from three areas: natural language interface and dialogue systems, database systems, and logic programming and deductive database systems. In this article, we shall place emphasis on cooperative databases.

We shall first provide an overview of cooperative database systems which covers such topics as presuppositions, misconceptions, intensional query answering, user modeling, query relaxation, and associative query answering. Then, we present the concept of the Type Abstraction Hierarchy (TAH) which provides a structured approach for query relaxation. Methodologies for automatic TAH generation are discussed. Next, we present the cooperative primitives for query relaxation and selected query examples for relational databases. Then, we present the relaxation controls for providing efficient query processing and the filtering of unsuitable answers for the user. The case-based approach for providing relevant information to query answers is then presented. The performance of a set of sample queries generated from an operational cooperative database system (CoBase) on top of a relational database is reported. Finally, we discuss the technology transfer of successful query relaxation to transportation, logistics planning applications, medical image databases, and electronic warfare applications.

OVERVIEW

Presuppositions

Usually when one asks a query, one not only presupposes the existence of all the components of the query, but one also presupposes an answer to the query itself. For example, suppose one asks "Which employees own red cars?" One assumes there is an answer to the query. If the answer is "nobody owns a red car," the system should provide the user with further explanation (e.g., in the case where no employee owns a red car because no employee owns a car at all). To avoid misleading the user, the answer should be "There are no employees who own a red car because no employee owns a car at all." Therefore in many queries, "No" as an answer does not provide the user with sufficient information. Further clarification is necessary to resolve the presupposition problem (2). False presuppositions usually occur with respect to the database's state and schema. Presuppositions assume that the query has an answer. If any presuppositions are false, the query is nonsensical. The following is a method to detect false presuppositions. Let us represent a query as a graph consisting of arcs at the nodes and binary relations between the arcs. The graph is a semantic network, and the query is reexpressed in binary notation. The query answering system checks to see that each connected subgraph is nonempty. If any is empty, this indicates a failed presupposition. A prototype system called CO-OP (A Cooperative Query System) was constructed and operated with a CODASYL database to demonstrate such cooperative concepts (3).

Misconceptions

A query may be free of any false presuppositions but can still cause misconceptions. False presuppositions concern the schema of the knowledge base. Misconceptions concern the scope of the domain of the knowledge base. Misconceptions arise when the user has a false or unclear understanding of what is necessarily true or false in the database. For example, for the query, "Which teachers take CS10?", the corresponding answer will be "None" followed by the explanation from the domain knowledge, "Teachers teach courses" and "Students take courses" (4). Whenever the user poses a query that has no answer, the system infers the probable mismatches between the user's view of the world and the knowledge in the knowledge base. The system then answers with a correction to rectify the mismatch (5).

Intensional Query Answering

Intensional query answering provides additional information about the extensional answer such as information about class hierarchies that define various data classes and relationships, integrity constraints to state the relationships among data, and rules that define new classes in terms of known classes. Intensional query answering can also provide abstraction and summarization of the extensional answer. As a result, the intensional answers can often improve and compliment extensional answers. For example, consider the query "Which cars are equipped with air bags?" The extensional answer will provide a very long list of registration numbers of all the cars that are equipped with air bags. However, an intensional answer will provide a summaried answer and state "All cars built after 1995 are equipped with air bags." Note that intensional answering gives more meaning to the answer than does the extensional answer. Further, intensional answers take less time to compute than extensional answers. There are different approaches to compute intensional query answers which yield different quality of answers (6–12). The effectiveness of the answer can be measured by completeness, nonredundancy, optimality, relevance, and efficiency (13).

User Models

Cooperative query answering depends on the user and context of the query. Thus, a user model will clearly aid in providing more specific query answering and thus improve search efficiency.

User models contain a representation of characteristic information about the user as well as a description of the user's intentions and goals. These models help interpret the content of a user's query and effectively customize results by guiding the query facility in deriving the answer.

Three types of knowledge about a user that are relevant to cooperative query answering are *interests and preferences, needs,* and *goals and intentions. Interests and preferences* direct the content and type of answers that should be provided. For example, (14) and (15) rewrite queries to include relevant information that is of interest to the user. *User needs* may vary from user to user. They can be represented by user constraints (16). The notion of user constraints is analogous to the integrity constraints in databases. Unlike integrity constraints, user constraints do not have to be logically consistent with the database. *Goals and intentions* do not vary from user to user. Rather, they vary from session to session and depend on the user who is attempting to achieve the goal. Past dialogue, user models, and other factors can help a system to determine the probable goals and intentions of the user (17–20) and also clarify the user's goals (21). The system can also explain the brief of the system that conflicts with the user's belief to resolve the user's misconceptions (22,23). Hemerly et al. (24) use a predefined user model and maintain a log of previous interactions to avoid misconstruction when providing additional information.

Query Relaxation

In conventional databases, if the required data is missing, if an exact answer is unavailable, or if a query is not well-formed with respect to the schema, the database just returns a null answer or an error. An intelligent system would be much more resourceful and cooperative by relaxing the query conditions and providing an approximate answer. Further, if the user does not know the exact database schema, the user is permitted to pose queries containing concepts that may not be expressed in the database schema.

A user interface for relational databases has been proposed (25) that is tolerant of incorrect user input and allows the user to select directions of relaxation. Chu, Cheng, and Lee (26) proposed to generalize queries by relaxing the query conditions via a knowledge structure called Type Abstraction Hierarchy (TAH). TAHs provide multilevel representation of domain knowledge. Relaxation can be performed via generalization and specialization (traversing up and down the hierarchy). Query conditions are relaxed to their semantic neighbors in the TAHs until the relaxed query conditions can produce approximate answers. Conceptual terms can be defined by labeling the nodes in a type abstraction hierarchy. To process a query with conceptual terms, the conceptual terms are translated into numeric value ranges or a set of nonnumeric information under that node. TAHs can then be generated by clustering algorithms from data sources. There are numerical TAHs that generate by clustering attributes with numerical databases (27,28) and nonnumerical TAHs that generate by rule induction from nonnumerical data sources (29).

Explicit relaxation operators such as approximate, near-to (distance range), and similar-to (based on the values of a set of attributes) can also be introduced in a query to relax the query conditions. Relaxation can be controlled by users with operators such as nonrelaxable, relaxation order, preference list, the number of answers, etc., which can be included in the query. A cooperative languge for relational databases, CoSQL, was developed (30,31) and extended the Structured Query Language (SQL) with these constructs. A cooperative database interface called CoBase was developed to automatically rewrite a CoSQL query with relaxation and relaxation control into SQL statements. As a result, CoBase can run on top of conventional relational databases such as Oracle, Sybase, etc., to provide query relaxation as well as conceptual query answering (answering to a query with conceptual terms) (27,31).

Gaasterland, Godfrey, and Minker (32) have used a similar type of abstraction knowledge representation for providing query relaxation in deductive databases by expanding the scope of query constraints. They also used a meta-interpreter to provide users with choices of relaxed queries.

Associative Query Answering

Associative Query Answering provides the user with additional useful relevant information about a query even if the user does not ask for or does not know how to ask for such information. Such relevant information can often expedite the query answering process or provide the user with additional topics for dialogue to accomplish a query goal. It can also provide valuable past experiences that may be helpful to the user in problem solving and decision making. For example, consider the query "Find an airport that can land a C5." In addition to the query answer regarding the location of the airport, additional relevant information for a pilot may be the *weather* and *runway conditions* of the airport. The additional relevant information for a transportation planner may be the existence of *railway facilities* and *storage facilities* nearby the airport. Thus associative information is both user- and context-sensitive. Cuppens and Demolombe (14) use a rule-based approach to rewrite queries by adding additional attributes to the query vector to provide additional relevant information. They defined a meta-level definition of a query, which specifies the query in three parts: entity, condition, and retrieved attributes. Answers to queries provide values to the variables designated by the retrieved attributes. They have defined methods to extend the retrieved attributes according to heuristics about topics of interest to the user.

CoBase uses a case-based reasoning approach to match past queries with the posed query (33). Query features consist of the query topic, the output attribute list, and the query conditions (15). The similarity of the query features can be evaluated from a user-specific semantic model based on the database schema, user type, and context. Cases with the same topic are searched first. If insufficient cases were found, then cases with related topics are searched. The attributes in the matched cases are then extended to the original query. The extended query is then processed to derive additional relevant information for the user.

STRUCTURED APPROACH FOR QUERY RELAXATION

Query relaxation relaxes a query scope to enlarge the search range or relaxes an answer scope to include additional information. Enlarging and shrinking a query scope can be accomplished by viewing the queried objects at different conceptual levels because an object representation has wider coverage at a higher level and, inversely, more narrow coverage at a lower level. We propose the notion of a type abstraction hierarchy (27–29) for providing an efficient and organized framework for cooperative query processing. A TAH represents objects at different levels of abstraction. For example, in Fig. 1, the Medium-Range (i.e., from 4000 to 8000 ft) in the TAH for runway length is a more abstract representation than a specific runway length in the same TAH (e.g., 6000 ft). Likewise, SW Tunisia is a more abstract representation than individual airports (e.g., Gafsa). A higher-level and more abstract object representation corresponds to multiple lower levels and more specialized object representations. Querying an abstractly represented object is equivalent to querying multiple specialized objects.

A query can be modified by relaxing the query conditions via such operations as generalization (moving up the TAH) and specialization (moving down the TAH, moving, for example, from 6000 ft to Medium-Range to (4000 ft, 8000 ft). In addition, queries may have conceptual conditions such as runway-length = Medium-Range. This condition can be transformed into specific query conditions by specialization. Query modification may also be specified explicitly by the user through a set of cooperative operators such as similar-to, approximate, and near-to.

The notion of multilevel object representation is not captured by the conventional semantic network and object-oriented database approaches for the following reasons. Grouping objects into a class and grouping several classes into a superclass provide only a common *title* (type) for the involved objects without concern for the object instance values and without introducing abstract object representations. Grouping several objects together and identifying their aggregation as a single (complex) object does not provide abstract instance representations for its component objects. Therefore, an object-oriented database deals with information only at two general layers: the metalayer and the instance layer. Because forming an object-oriented type hierarchy does not introduce new instance values, it is impossible to introduce an additional instance layer. In the TAH, instances of a supertype and a subtype may have different representations and can be viewed at different instance layers. Such multiple-layer knowledge representation is essential for cooperative query answering.

Knowledge for query relaxation can be expressed as a set of logical rules, but such a rule-based approach (14) lacks a systematic organization to guide the query transformation process. TAHs provide a much simpler and more intuitive representation for query relaxation and do not have the complexity of the inference that exists in the rule-based system. As a result, the TAH structure can easily support flexible relaxation control (see subsection entitled "Relaxation Agent with CORBA Interface"), which is important to improve relaxation accuracy and efficiency. Furthermore, knowledge represented in a TAH is customized; thus changes in one TAH represent only a localized update and do not affect other TAHs, simplifying TAH maintenance (see subsection entitled "Maintenance of TAHs"). We have developed tools to generate TAHs

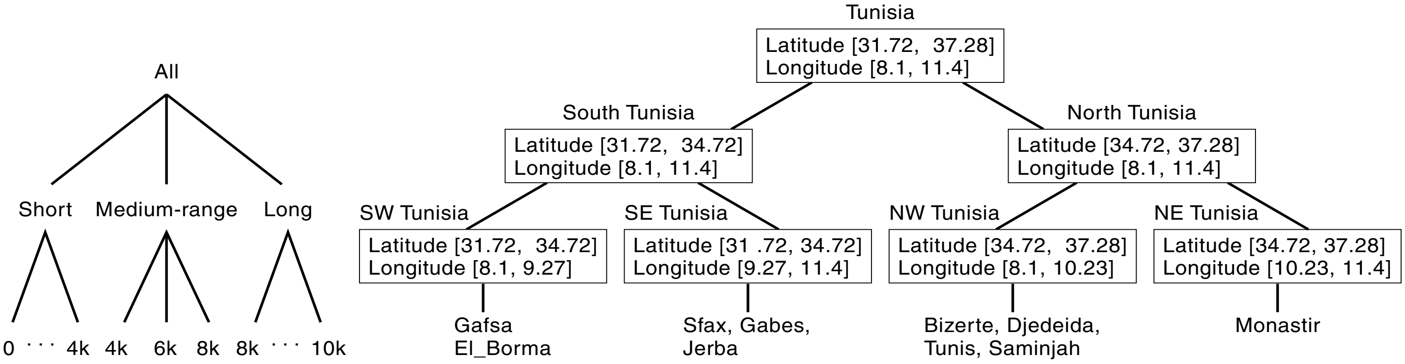

Figure 1. Type abstraction hierarchies: (a) runway length and (b) airport location in Tunisia.

automatically from data sources (see the next section), which enable our system to scale up and extend to large data sources.

AUTOMATIC KNOWLEDGE ACQUISITION

The automatic generation of a knowledge base (TAHs) from databases is essential for CoBase to be scalable to large systems. We have developed algorithms to generate automatically TAHs based on database instances. A brief discussion about the algorithms and their complexity follow.

Numerical TAHs

COBWEB (34), a conceptual clustering system, uses category utility (35) as a quality measure to classify the objects described by a set of attributes into a classification tree. COBWEB deals only with categorical data. Thus, it cannot be used for abstracting numerical data. For providing approximate answers, we want to build a classification tree that minimizes the difference between the desired answer and the derived answer. Specifically, we use relaxation error as a measure for clustering. The relaxation error (RE) is defined as the average difference between the requested values and the returned values. $\mathrm{RE}_1(C)$ can also be interpreted from the standpoint of query relaxation. Let us define the relaxation error of x_i, $\mathrm{RE}_1(x_i)$, as the average difference from x_i to x_j, $j = 1, \ldots, n$. That is,

$$\mathrm{RE}_1(x_i) = \sum_{j=1}^{n} P(x_j)|x_i - x_j| \tag{1}$$

where $P(x_j)$ is the occurrence probability of x_j in C. $\mathrm{RE}_1(x_i)$ can be used to measure the quality of an approximate answer where x_i in a query is relaxed to x_j, $j = 1, \ldots, n$. Summing $\mathrm{RE}_1(x_i)$ over all values x_i in C, we have

$$\mathrm{RE}_1(C) = \sum_{i=1}^{n} P(x_i)\mathrm{RE}_1(x_i) \tag{2}$$

Thus, $\mathrm{RE}_1(C)$ is the expected error of relaxing any value in C.

If $\mathrm{RE}_1(C)$ is large, query relaxation based on C may produce very poor approximate answers. To overcome this problem, we can partition C into subclusters to reduce relaxation error. Given a partition $P = \{C_1, C_2, \ldots, C_N\}$ of C, the relaxation error of the partition P is defined as

$$\mathrm{RE}_1(P) = \sum_{k=1}^{N} P(C_k)\mathrm{RE}_1(C_k) \tag{3}$$

where $P(C_k)$ equals the number of tuples in C_k divided by the number of tuples in C. In general, $\mathrm{RE}_1(P) < \mathrm{RE}_1(C)$.

Relaxation error is the expected pairwise difference between values in a cluster. The notion of relaxation error for multiple attributes can be extended from single attributes.

DIstribution Sensitive Clustering (DISC) (27,28) partitions sets of numerical values into clusters that minimize the relaxation error. We shall now present a class of DISC algorithms for clustering numerical values. We shall present the algorithm for a single attribute and then extend it for multiple attributes.

The Clustering Algorithm for a Single Attribute. Given a cluster with n distinct values, the number of partitions is exponential with respect to n, so the best partition takes exponential time to find. To reduce computation complexity, we shall consider only binary partitions. Later we shall show that a simple hill-climbing strategy can be used for obtaining N-ary partitions from binary partitions.

Our method is top down: we start from one cluster consisting of all the values of an attribute, and then we find *cuts* to partition recursively the cluster into smaller clusters. (A *cut* c is a value that separates a cluster of numbers $\{x|a \le x \le b\}$ into two subclusters $\{x|a \le x \le c\}$ and $\{x|c < x \le b\}$.) The partition result is a concept hierarchy called type abstraction hierarchy. The clustering algorithm is called the DISC method and is given in Table 1.

In Ref. 30, an implementation of the algorithm BinaryCut is presented whose time complexity is $O(n)$. Because DISC needs to execute BinaryCut $n - 1$ times at most to generate a TAH, the worst case time complexity of DISC is $O(n^2)$. [The average case time complexity of DISC is $O(n \log n)$.]

***N*-ary Partitioning.** N-ary partitions can be obtained from binary partitions by a hill-climbing method. Starting from a binary partition, the subcluster with greater relaxation error is selected for further cutting. We shall use RE as a measure to determine if the newly formed partition is better than the previous one. If the RE of the binary partition is less than that of the trinary partition, then the trinary partition is dropped, and the cutting is terminated. Otherwise, the trinary partition is selected, and the cutting process continues until it reaches the point where a cut increases RE.

The Clustering Algorithm for Multiple Attributes. Query relaxation for multiple attributes using multiple single-attribute TAHs relaxes each attribute independently disregarding the relationships that might exist among attributes. This may not be adequate for the applications where attributes are dependent. (Dependency here means that all the attributes as a whole define a coherent concept. For example, the length and width of a rectangle are said to be "semantically" dependent. This kind of dependency should be distinguished from the functional dependency in database theory.) In addition, using multiple single-attribute TAHs is inefficient because it may need many iterations of query modification and database access before approximate answers are found. Furthermore, relaxation control for multiple TAHs is more complex because there is a large number of possible orders for relaxing attributes. In general, we can rely only on simple heuristics such as best first or minimal coverage first to guide the relaxation (see subsection entitled "Relaxation Control"). These heuristics cannot guarantee best approximate answers because they are rules of thumb and not necessarily accurate.

Most of these difficulties can be overcome by using Multiattribute TAH (MTAH) for the relaxation of multiple attributes. Because MTAHs are generated from semantically dependent attributes, these attributes are relaxed together in a single relaxation step, thus greatly reducing the number of query modifications and database accesses. Approximate answers derived by using MTAH have better quality than those derived by using multiple single-attribute TAHs. MTAHs are context- and user-sensitive because a user may generate several MTAHs with different attribute sets from a table. Should

Table 1. The Algorithms DISC and BinaryCut

```
Algorithm DISC(C)
    if the number of distinct values ∈ C < T, return /* T is a threshold */
    let cut = the best cut returned by BinaryCut(C)
    partition values in C based on cut
    let the resultant subclusters be C1 and C2
        call DISC(C1) and DISC(C2)

Algorithm BinaryCut(C)
/* input cluster C = {x1, . . ., xn} */
    for h = 1 to n − 1 /* evaluate each cut */
        Let P be the partition with clusters C1 = {x1, . . ., xh} and C2 = {xh+1, . . ., xn}
        compute RE1(P)
        if RE1(P) < MinRE then
            MinRE = RE1(P) , cut = h /* the best cut */
    Return cut as the best cut
```

a user need to create an MTAH containing semantically dependent attributes from different tables, these tables can be joined into a single view for MTAH generation.

To cluster objects with multiple attributes, DISC can be extended to Multiple attributes–DISC or M-DISC (28). MTAHs are generated. The algorithm DISC is a special case of M-DISC, and TAH is a special case of MTAH. Let us now consider the time complexity of M-DISC. Let m be the number of attributes and n be the number of distinct attribute values. The computation of relaxation error for a single attribute takes $O(n \log n)$ to complete (27). Because the computation of RE involves computation of relaxation error for m attributes, its complexity is $O(mn \log n)$. The nested loop in M-DISC is executed mn times so that the time complexity of M-DISC is $O(m^2n^2 \log n)$. To generate an MTAH, it takes no more than n calls of M-DISC; therefore, the worst case time complexity of generating an MTAH is $O(m^2n^3 \log n)$. The average case time complexity is $O[m^2n^2(\log n)^2]$ because M-DISC needs only to be called $\log n$ times on the average.

Nonnumerical TAHs

Previous knowledge discovery techniques are inadequate for clustering nonnumerical attribute values for generating TAHs for Cooperative Query Answering. For example, Attribute Oriented Induction (36) provides summary information and characterizes tuples in the database, but is inappropriate since attribute values are focused too closely on a specific target. Conceptual Clustering (37,38) is a top-down method to provide approximate query answers, iteratively subdividing the tuple-space into smaller sets. The top-down approach does not yield clusters that provide the best correlation near the bottom of the hierarchy. Cooperative query answering operates from the bottom of the hierarchy, so better clustering near the bottom is desirable. To remedy these shortcomings, a bottom-up approach for constructing attribute abstraction hierarchies called Pattern-Based Knowledge Induction (PKI) was developed to include a nearness measure for the clusters (29).

PKI determines clusters by deriving rules from the instance of the current database. The rules are not 100% certain; instead, they are rules-of-thumb about the database, such as

If the car is a sports car, then the color is red

Each rule has a *coverage* that measures how often the rule applies, and *confidence* measures the validity of the rule in the database. In certain cases, combining simpler rules can derive a more sophisticated rule with high confidence.

The PKI approach generates a set of useful rules that can then be used to construct the TAH by clustering the premises of rules sharing a similar consequence. For example, if the following two rules:

If the car is a sports car, then the color is red

If the car is a sports car, then the color is black

have high confidence, then this indicates that for sports cars, the colors *red* and *black* should be clustered together. Supporting and contradicting evidence from rules for other attributes is gathered and PKI builds an initial set of clusters. Each invocation of the clustering algorithm adds a layer of abstraction to the hierarchy. Thus, attribute values are clustered if they are used as the premise for rules with the same consequence. By iteratively applying the algorithm, a hierarchy of clusters (TAH) can be found. PKI can cluster attribute values with or without expert direction. The algorithm can be improved by allowing domain expert supervision during the clustering process. PKI also works well when there are NULL values in the data. Our experimental results confirm that the method is scalable to large systems. For a more detailed discussion, see (29).

Maintenance of TAHs

Because the quality of TAH affects the quality of derived approximate answers, TAHs should be kept up to date. One simple way to maintain TAHs is to regenerate them whenever an update occurs. This approach is not desirable because it causes overhead for the database system. Although each update changes the distribution of data (thus changing the quality of the corresponding TAHs), this may not be significant enough to warrant a TAH regeneration. TAH regeneration is necessary only when the cumulative effect of updates has greatly degraded the TAHs. The quality of a TAH can be monitored by comparing the derived approximate answers to the expected relaxation error (e.g., see Fig. 7), which is computed at TAH generation time and recorded at each node of the TAH. When the derived approximate answers significantly deviate from the expected quality, then the quality of the

TAH is deemed to be inadequate and a regeneration is necessary. The following incremental TAH regeneration procedure can be used. First, identify the node within the TAH that has the worst query relaxations. Apply partial TAH regeneration for all the database instances covered by the node. After several such partial regenerations, we then initiate a complete TAH regeneration.

The generated TAHs are stored in UNIX files, and a TAH Manager (described in subsection entitled "TAH Facility") is responsible to parse the files, create internal representation of TAHs, and provide operations such as generalization and specialization to traverse TAHs. The TAH Manager also provides a directory that describes the characteristics of TAHs (e.g., attributes, names, user type, context, TAH size, location) for the users/systems to select the appropriate TAH to be used for relaxation.

Our experience in using DISC/M-DISC and PKI for ARPA Rome Labs Planning Initiative (ARPI) transportation databases (94 relations, the biggest one of which has 12 attributes and 195,598 tuples) shows that the clustering techniques for both numerical and nonnumerical attributes can be generated from a few seconds to a few minutes depending on the table size on a Sun SPARC 20 Workstation.

COOPERATIVE OPERATIONS

The cooperative operations consist of the following four types: context-free, context-sensitive, control, and interactive.

Context-Free Operators

- *Approximate operator* $\wedge v$ relaxes the specified value v within the approximate range predefined by the user. For example, `^9am` transforms into the interval (8am, 10am).
- *Between* (v_1, v_2) specifies the interval for an attribute. For example, time `between (7am, ^9am)` transforms into (7am, 10am). The transformed interval is prespecified by either the user or the system.

Context-Sensitive Operators

- *Near-to X* is used for specification of spatial nearness of object X. The near-to measure is context- and user-sensitive. "Nearness" can be specified by the user. For example, `near-to 'BIZERTE'` requests the list of cities located within a certain Euclidean distance (depending on the context) from the city Bizerte.
- *Similar-to X based-on* $[(a_1\ w_1)(a_2\ w_2) \cdots (a_n\ w_n)]$ is used to specify a set of objects semantically similar to the target object X based on a set of attributes $(a_1, a_2, \ldots, a_n)$ specified by the user. Weights $(w_1, w_2, \ldots, w_n)$ may be assigned to each of the attributes to reflect the relative importance in considering the similarity measure. The set of similar objects can be ranked by the similarity. The similarity measures that computed from the nearness (e.g., weighted mean square error) of the prespecified attributes to that of the target object. The set size is bound by a prespecified nearness threshold.

Control Operators

- *Relaxation-order* $(a_1, a_2, \ldots, a_n)$ specifies the order of the relaxation among the attributes $(a_1, a_2, \ldots, a_n)$ (i.e., a_i precedes a_{i+1}). For example, `relaxation-order (runway_length, runway_width)` indicates that if no exact answer is found, then runway_length should be relaxed first. If still no answer is found, then relax the runway_width. If no relaxation-order control is specified, the system relaxes according to its default relaxation strategy.
- *Not-relaxable* $(a_1, a_2, \ldots, a_n)$ specifies the attributes $(a_1, a_2, \ldots, a_n)$ that should not be relaxed. For example, `not-relaxable location_name` indicates that the condition clause containing location_name must not be relaxed.
- *Preference-list* $(v_1, v_2, \ldots, v_n)$ specifies the preferred values $(v_1, v_2, \ldots, v_n)$ of a given attribute, where v_i is preferred over v_{i+1}. As a result, the given attribute is relaxed according to the order of preference that the user specifies in the preference list. Consider the attribute "food style"; a user may prefer Italian food to Mexican food. If there are no such restaurants within the specified area, the query can be relaxed to include the foods similar to Italian food first and then similar to Mexican food.
- *Unacceptable-list* $(v_1, v_2, \ldots, v_n)$ allows users to inform the system not to provide certain answers. This control can be accomplished by trimming parts of the TAH from searching. For example, `avoid airlines X and Y` tells the system that airlines X and Y should not be considered during relaxation. It not only provides more satisfactory answers to users but also reduces search time.
- *Alternative-TAH (TAH-name)* allows users to use the TAHs of their choices. For example, a vacation traveler may want to find an airline based on its fare, whereas a business traveler is more concerned with his schedule. To satisfy the different needs of the users, several TAHs of airlines can be generated, emphasizing different attributes (e.g., price and nonstop flight).
- *Relaxation-level* (v) specifies the maximum allowable range of the relaxation on an attribute, i.e., $[0, v]$.
- *Answer-set* (s) specifies the minimum number of answers required by the user. CoBase relaxes query conditions until enough number of approximate answers (i.e., $\geq s$) are obtained.
- *Rank-by* $((a_1, w_1), (a_2, w_2), \ldots, (a_n, w_n))$ *METHOD* $(method - name)$ specifies a method to rank the answers returned by CoBase.

User/System Interaction Operators

- *Nearer, Further* provide users with the ability to control the near-to relaxation scope interactively. Nearer reduces the distance by a prespecified percentage, whereas further increases the distance by a prespecified percentage.

Approximate operator relaxation range

Relation name	Attribute name	∧Range
Aports	Runway_length_ft	500
Aports	Runway_width_ft	10
Aports	Parking_sq_ft	100000
GEOLOC	Latitude	0.001
GEOLOC	Longitude	0.001

Near-to operator relaxation range

Relation name	Attribute name	Near-to range	Nearer/further
Aports	Aport_name	100 miles	50%
GEOLOC	Location_name	200 miles	50%

Figure 2. Relaxation range for the approximate and near-to operators.

Editing Relaxation Control Parameters

Users can browse and edit relaxation control parameters to better suit their applications (see Fig. 2). The parameters include the relaxation range for the approximately-equal operator, the default distance for the near-to operator, and the number of returned tuples for the similar-to operator.

Cooperative SQL (CoSQL)

The cooperative operations can be extended to the relational database query language, SQL, as follows: The context-free and context-sensitive cooperative operators can be used in conjunction with attribute values specified in the WHERE clause. The relaxation control operators can be used only on attributes specified in the WHERE clause, and the control operators must be specified in the WITH clause after the WHERE clause. The interactive operators can be used alone as command inputs.

Examples. In this section, we present a few selected examples that illustrate the capabilities of the cooperative operators. The corresponding TAHs used for query modification are shown in Fig. 1, and the relaxable ranges are shown in Fig. 2.

Query 1. List all the airports with the runway length greater than 7500 ft and runway width greater than 100 ft. If there is no answer, relax the runway length condition first. The following is the corresponding CoSQL query:

```
SELECT aport_name, runway_length_ft,
  runway_width_ft
FROM aports
WHERE runway_length_ft > 7500 AND
  runway_width_ft > 100
WITH RELAXATION-ORDER (runway_length_ft,
  runway_width_ft)
```

Based on the TAH on runway length and the relaxation order, the query is relaxed to

```
SELECT aport_name, runway_length_ft,
  runway_width_ft
FROM aports
WHERE runway_length_ft >= 7000 AND
  runway_width_ft > 100
```

If this query yields no answer, then we proceed to relax the range runway width.

Query 2. Find all the cities with their geographical coordinates near the city Bizerte in the country Tunisia. If there is no answer, the restriction on the country should not be relaxed. The near-to range in this case is prespecified at 100 miles. The corresponding CoSQL query is as follows:

```
SELECT location_name, latitude, longitude
FROM GEOLOC
WHERE location_name NEAR-TO 'Bizerte'
  AND country_state_name = 'Tunisia'
WITH NOT-RELAXABLE country_state_name
```

Based on the TAH on location Tunisia, the relaxed version of the query is

```
SELECT location_name, latitude, longitude
FROM GEOLOC
WHERE location_name IN {'Bizerte', 'Djedeida',
  'Gafsa',
      'Gabes', 'Sfax', 'Sousse', 'Tabarqa',
        'Tunis'}
    AND country_state_name_= 'Tunisia'
```

Query 3. Find all airports in Tunisia similar to the Bizerte airport. Use the attributes runway_length_ft and runway_ width_ft as criteria for similarity. Place more similarity emphasis on runway length than runway width; their corresponding weight assignments are 2 and 1, respectively. The following is the CoSQL version of the query:

```
SELECT aport_name
FROM aports, GEOLOC
WHERE aport_name SIMILAR-TO 'Bizerte'
      BASED-ON ((runway_length_ft 2.0)
        (runway_width_ft 1.0))
  AND country_state_name = 'TUNISIA'
  AND GEOLOC.geo_code = aports.geo_code
```

To select the set of the airport names that have the runway length and runway width similar to the ones for the airport in Bizerte, we shall first find all the airports in Tunisia and, therefore, transform the query to

```
SELECT aport_name
FROM aports, GEOLOC
WHERE country_state_name_ = 'TUNISIA'
  AND GEOLOC.geo_code = aports.geo_code
```

After retrieving all the airports in Tunisia, based on the runway length, runway width, and their corresponding weights,

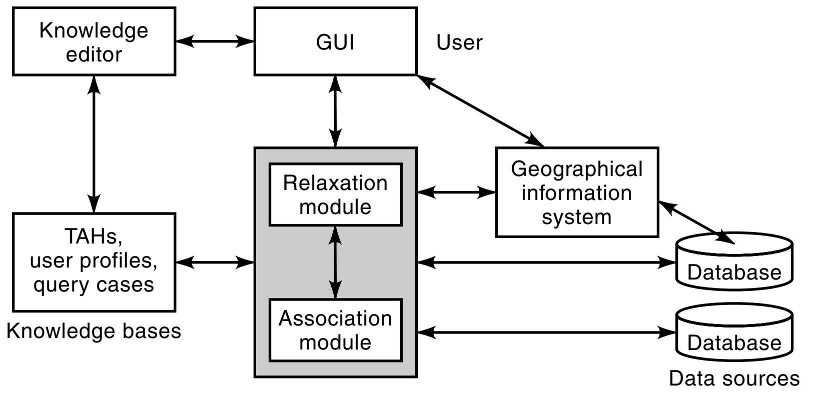

Figure 3. CoBase functional architecture.

the similarity of these airports to Bizerte can be computed by the prespecified nearness formula (e.g., weighted mean square error). The order in the similarity set is ranked according to the nearness measure, and the size of the similarity set is determined by the prespecified nearness threshold.

A SCALABLE AND EXTENSIBLE ARCHITECTURE

Figure 3 shows an overview of the CoBase System. Type abstraction hierarchies and relaxation ranges for the explicit operators are stored in a knowledge base (KB). There is a TAH directory storing the characteristics of all the TAHs in the system. When CoBase queries, it asks the underlying database systems (DBMS). When an approximate answer is returned, context-based semantic nearness will be provided to rank the approximate answers (in order of nearness) against the specified query. A graphical user interface (GUI) displays the query, results, TAHs, and relaxation processes. Based on user type and query context, associative information is derived from past query cases. A user can construct TAHs from one or more attributes and modify the existing TAH in the KB.

Figure 4 displays the various cooperative modules: Relaxation, Association, and Directory. These agents are connected selectively to meet applications' needs. An application that requires relaxation and association capabilities, for example, will entail a linking of Relaxation and Association agents. Our architecture allows incremental growth with application. When the demand for certain modules increases, additional copies of the modules can be added to reduce the loading; thus, the system is scalable. For example, there are multiple copies of relaxation agent and association agent in Fig. 4. Further, different types of agents can be interconnected and communicate with each other via a common communication protocol [e.g., CORBA/IIOP, or Knowledge Query Manipulation Language (KQML) (39)] to perform a joint task. Thus, the architecture is extensible.

Relaxation Module

Query relaxation is the process of understanding the semantic context, intent of a user query and modifying the query constraints with the guidance of the customized knowledge structure (TAH) into near values that provide best-fit answers. The flow of the relaxation process is depicted in Fig. 5. When a CoSQL query is presented to the Relaxation Agent, the system first go through a preprocessing phase. During the preprocessing, the system first relaxes any context-free and/or context-sensitive cooperative operators in the query. All relaxation control operations specified in the query will be processed. The information will be stored in the relaxation manager and be ready to be used if the query requires relaxation. The modified SQL query is then presented to the underlying database system for execution. If no answers are returned, then the cooperative query system, under the direction of the

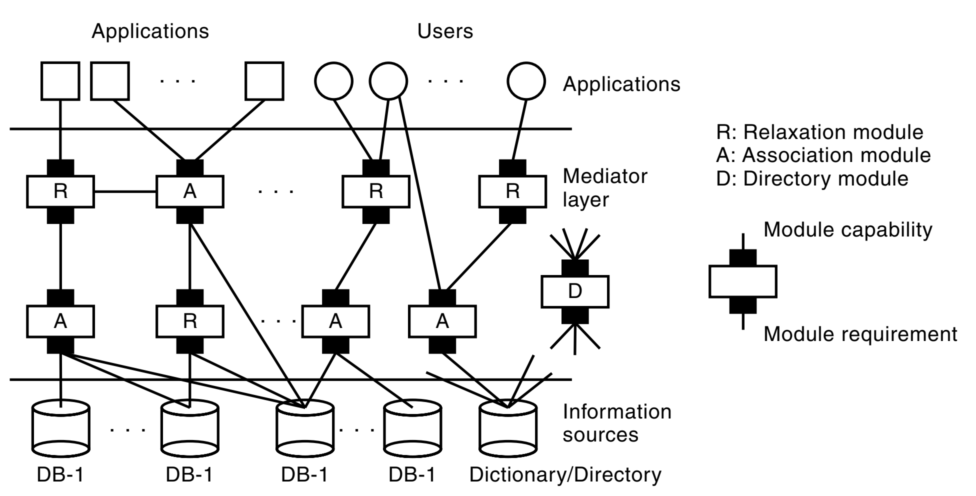

Figure 4. A scalable and extensible cooperative information system.

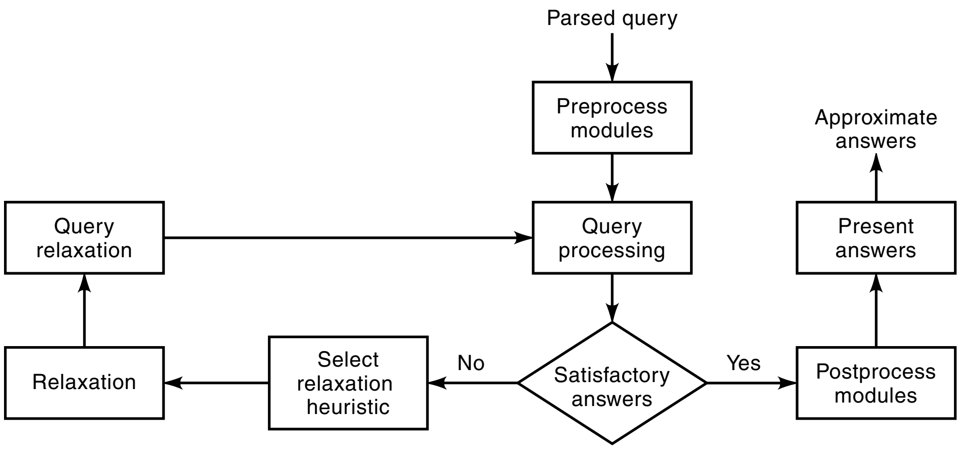

Figure 5. Flow chart for processing CoBase queries.

Relaxation Manager, relaxes the queries by query modification. This is accomplished by traversing along the TAH node for performing generalization and specialization and rewriting the query to include a larger search scope. The relaxed query is then executed, and if there is no answer, we repeat the relaxation process until we obtain one or more approximate answers. If the system fails to produce an answer due to overtrimmed TAHs, the relaxation manager will deactivate certain relaxation rules to restore part of a trimmed TAH to broaden the search scope until answers are found. Finally, the answers are postprocessed (e.g., ranking and filtering).

Relaxation Control. Relaxation without control may generate more approximations than the user can handle. The policy for relaxation control depends on many factors, including user profile, query context, and relaxation control operators as defined previously. The Relaxation Manager combines those factors via certain policies (e.g., minimizing search time or nearness) to restrict the search for approximate answers. We allow the input query to be annotated with control operators to help guide the agent in query relaxation operations.

If control operators are used, the Relaxation Manager selects the condition to relax in accordance with the requirements specified by the operators. For example, a relaxation-order operator will dictate "relax location first, then runway length." Without such user-specified requirements, the Relaxation Manager uses a default relaxation strategy by selecting the relaxation order based on the minimum coverage rule. Coverage is defined as the ratio of the cardinality of the set of instances covered by the entire TAH. Thus, coverage of a TAH node is the percentage of all tuples in the TAH covered by the current TAH node. The minimum coverage rule always relaxes the condition that causes the minimum increase in the scope of the query, which is measured by the coverage of its TAH node. This default relaxation strategy attempts to add the smallest number of tuples possible at each step, based on the rationale that the smallest increase in scope is likely to generate the close approximate answers. The strategy for choosing which condition to be relaxed first is only one of many possible relaxation strategies; the Relaxation Manager can support other different relaxation strategies as well.

Let us consider the following example of using control operators to improve the relaxation process. Suppose a pilot is searching for an airport with an 8000 ft runway in Bizerte but there is no airport in Bizerte that meets the specifications. There are many ways to relax the query in terms of location and runway length. If the pilot specifies the relaxation order to relax the location attribute first, then the query modification generalizes the location Bizerte to NW Tunisia (as shown in Fig. 1) and specifies the locations Bizerte, Djedeida, Tunis, and Saminjah, thus broadening the search scope of the original query. If, in addition, we know that the user is interested only in the airports in West Tunisia and does not wish to shorten the required runway length, the system can eliminate the search in East Tunisia and also avoid airports with short and medium runways, as shown in Fig. 6. As a result, we can limit the query relaxation to a narrower scope by trimming the TAHs, thus improving both the system performance and the answer relevance.

Spatial Relaxation and Approximation. In geographical queries, spatial operators such as located, within, contain, intersect, union, and difference are used. When there are no exact answers for a geographical query, both its spatial and nonspatial conditions can be relaxed to obtain the approximate answers. CoBase operators also can be used for describing approximate spatial relationships. For example, "an aircraft-carrier is *near* seaport Sfax." Approximate spatial operators, such as near-to and between are developed for the approximate spatial relationships. Spatial approximation depends on contexts and domains (40,41). For example, a hospital near to LAX is different from an airport near to LAX. Likewise, the nearness of a hospital in a metropolitan area is different from the one in a rural area. Thus, spatial conditions should be relaxed differently in different circumstances. A common approach to this problem is the use of prespecified ranges. This approach requires experts to provide such information for all possible situations, which is difficult to scale up to larger applications or to extend to different domains. Because TAHs are user- and context-sensitive, they can be used to provide context-sensitive approximation. More specifically, we can generate TAHs based on multidimensional spatial attributes (MTAHs).

Further, MTAH (based on latitude and longitude) is generated based on the distribution of the object locations. The distance between nearby objects is context-sensitive: the denser the location distribution, the smaller the distance among the objects. In Fig. 7, for example, the default neighborhood dis-

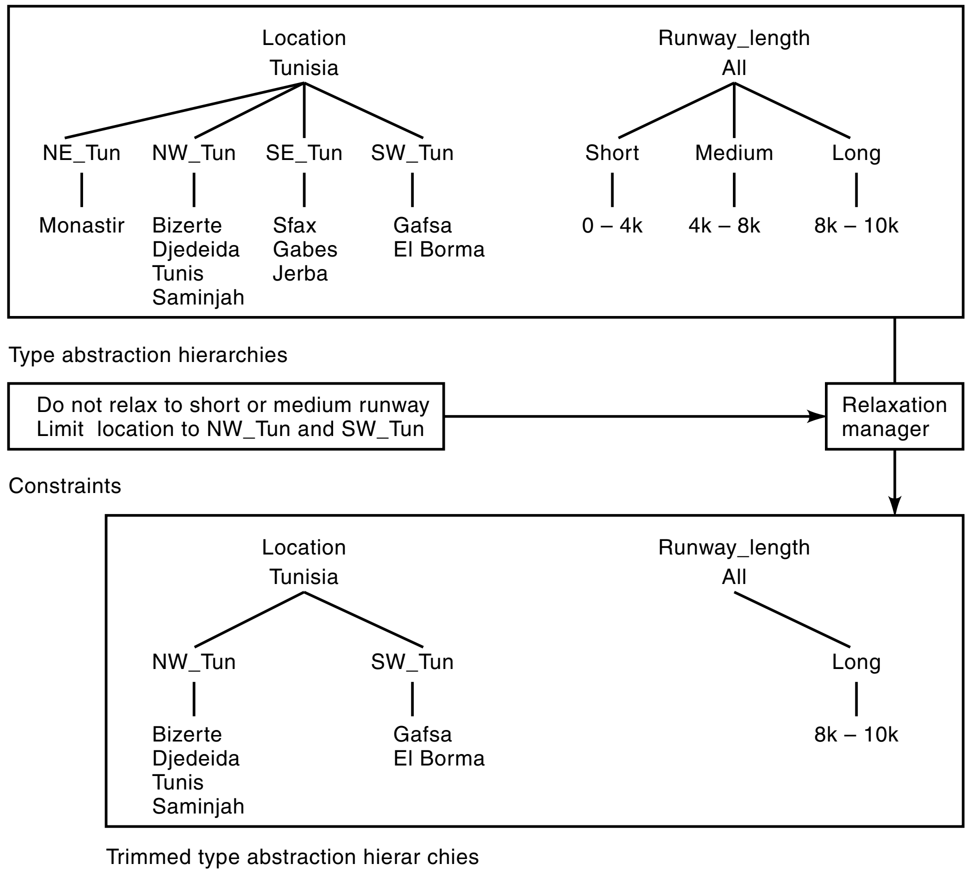

Figure 6. TAH trimming based on relaxation control operators.

tance in Area 3 is smaller than the one in Area 1. Thus, when a set of airports is clustered based on the locations of the airports, the ones in the same cluster of the MTAH are much closer to each other than to those outside the cluster. Thus, they can be considered near-to each other. We can apply the same approach to other approximate spatial operators, such as between (i.e., a cluster near-to the center of two objects). MTAHs also can be used to provide context-sensitive query relaxation. For example, consider the query: "Find an airfield at the city Sousse." Because there is no airfield located exactly at Sousse, this query can be relaxed to obtain approximate answers. First, we locate the city Sousse with latitude 35.83 and longitude 10.63. Using the MTAH in Fig. 7, we find that Sousse is covered by Area 4. Thus, the airport Monastir is returned. Unfortunately, it is not an airfield. So the query is further relaxed to the neighboring cluster—the four airports in Area 3 are returned: Bizerte, Djedeida, Tunis, and Saminjah. Because only Djedeida and Saminjah are airfields, these two will be returned as the approximate answers.

MTAHs are automatically generated from databases by using our clustering method that minimizes relaxation error (27). They can be constructed for different contexts and user type. For example, it is critical to distinguish a friendly airport from an enemy airport. Using an MTAH for friendly air-

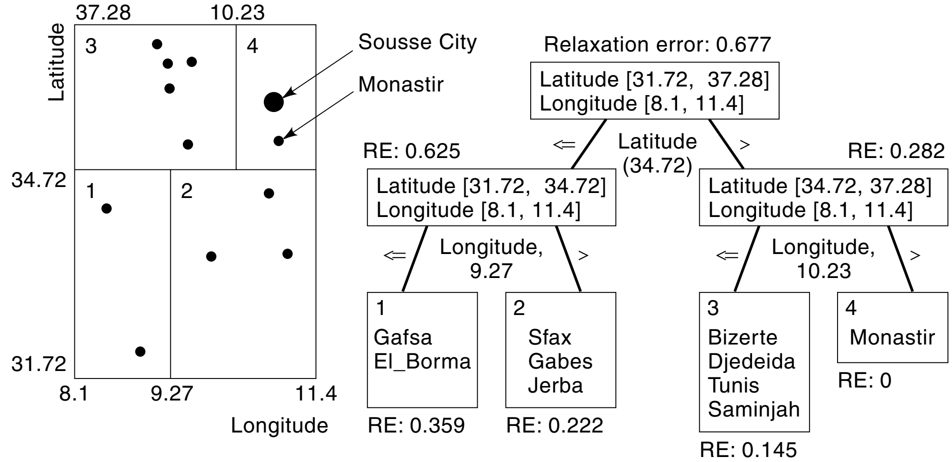

Figure 7. An MTAH for the airports in Tunisia and its corresponding two-dimensional space.

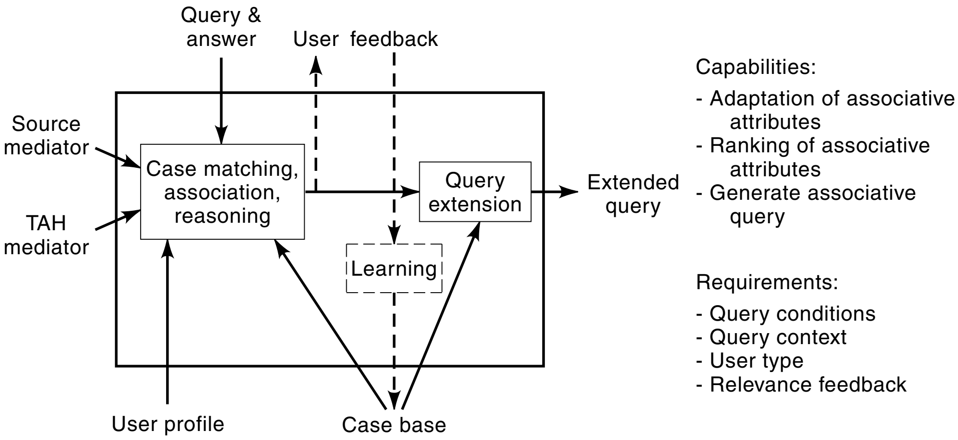

Figure 8. Associative query answering facility.

ports restricts the relaxation only within the set of friendly airports, even though some enemy airports are geographically nearby. This restriction significantly improves the accuracy and flexibility of spatial query answering. The integration of spatial and cooperative operators provides more expressiveness and context-sensitive answers. For example, the user is able to pose such queries as, "find the airports similar-to LAX and near-to City *X*." When no answers are available, both near-to and similar-to can be relaxed based on the user's preference (i.e., a set of attributes). To relax near-to, airports from neighboring clusters in the MTAH are returned. To relax similar-to, the multiple-attribute criteria are relaxed by their respective TAHs.

Cooperativeness in geographic databases was studied in Ref. 42. A rule-based approach is used in their system for approximate spatial operators as well as query relaxation. For example, they define that "P is near-to Q iff the distance from P to Q is less than *n*length_unit,* where *length_unit* is a context dependent scalar parameter, and *n* is a scalar parameter that can be either unique for the application and thus defined in domain model, or specific for each class of users and therefore defined in the user models." This approach requires *n* and *length_unit* be set by domain experts. Thus, it is difficult to scale up. Our system uses MTAHs as a representation of the domain knowledge. The MTAHs can be generated automatically from databases based on contexts and provide a structured and context-sensitive way to relax queries. As a result, it is scalable to large applications. Further, the relaxation error at each node is computed during the construction of TAHs and MTAHs. It can be used to evaluate the quality of relaxations and to rank the nearness of the approximate answers to the exact answer.

Associative Query Answering via Case-Based Reasoning

Often it is desirable to provide additional information relevant to, though not explicitly stated in, a user's query. For example, in finding the location of an airport satisfying the runway length and width specifications, the association module (Fig. 8) can provide additional information about the runway quality and weather condition so that this additional information may help the pilot select a suitable airport to land his aircraft. On the other hand, the useful relevant information for the same query if posed by a transportation planner may be information regarding railway facilities and storage facilities nearby the airport. Therefore, associative information is user- and context-sensitive.

Association in CoBase is executed as a multistep postprocess. After the query is executed, the answer set is gathered with the query conditions, user profile, and application constraints. This combined information is matched against query cases from the case base to identify relevant associative information (15,33). The query cases can take the form of a CoBase query, which can include any CoBase construct, such as conceptual conditions (e.g., runway_length_ft = short) or explicitly cooperative operations (city near-to 'BIZERTE').

For example, consider the query

```
SELECT name, runway_length_ft
FROM airports
WHERE runway_length_ft > 6000
```

Based on the combined information, associative attributes such as runway conditions and weather are derived. The associated information for the corresponding airports is retrieved from the database and then appended to the query answer, as shown in Fig. 9.

Our current case base, consisting of about 1500 past queries, serves as the knowledge server for the association module. The size of the case base is around 2 Mbyte. For association purposes, we use the 300-case set, which is composed of past queries used in the transportation domain. For testing performance and scalability of the system, we use a 1500-case set, which consists of randomly generated queries based on user profile and query template over the transportation domain. Users can also browse and edit association control

Query answer		Associative information	
Name	Runway_length	Runway_condition	Weather
Jerba	9500	Damaged	Sunny
Monastir	6500	Good	Foggy
Tunis	8500	Good	Good

Figure 9. Query answer and associative information for the selected airports.

parameters such as the number of association subjects, the associated links and weights of a given case, and the threshold for association relevance.

PERFORMANCE EVALUATION

In this section, we present the CoBase performance based on measuring the execution of a set of queries on the CoBase testbed developed at UCLA for the ARPI transportation domain. The performance measure includes response time for query relaxation, association, and the quality of answers. The response time depends on the type of queries (e.g., size of joins, number of joins) as well as the amount of relaxation, and association, required to produce an answer. The quality of the answer depends on the amount of relaxation and association involved. The user is able to specify the relaxation and association control to reduce the response time and also to specify the requirement of answer accuracy. In the following, we shall show four example queries and their performances. The first query illustrates the relaxation cost. The second query shows the additional translation cost for the "similar-to" cooperative operator, whereas the third query shows the additional association cost. The fourth query shows the processing cost for returned query answers as well as the quality of answers by using TAH versus MTAH for a very large database table (about 200,000 tuples).

Query 4. Find nearby airports can land C-5.

Based on the airplane location, the relaxation module translates *nearby* to a prespecified or user-specified latitude and longitude range. Based on the domain knowledge of C-5, the mediator also translates *land* into required runway length and width for landing the aircraft. The system executes the translated query. If no airport is found, the system relaxes the distance (by a predefined amount) until an answer is returned. In this query, an airport is found after one relaxation. Thus, two database retrievals (i.e., one for the original query and one for the relaxed query) are performed. Three tables are involved: Table GEOLOC (50,000 tuples), table RUNWAYS (10 tuples), and table AIRCRAFT_AIRFIELD_CHARS (29 tuples). The query answers provide airport locations and their characteristics.

```
Elapsed time: 5 seconds processing time for
               relaxation
              40 seconds database retrieval
               time
```

Query 5. Find at least three airports similar-to Bizerte based on runway length and runway width.

The relaxation module retrieves runway characteristics of Bizerte airport and translates the similar-to condition into the corresponding query conditions (runway length and runway width). The system executes the translated query and relaxes the runway length and runway width according to the TAHs until at least three answers are returned. Note that the TAH used for this query is a Runway-TAH based on runway length and runway width, which is different from the Location-TAH based on latitude and longitude (shown in Fig. 7). The nearness measure is calculated based on weighted mean square error. The system computes similarity measure for each answer obtained, ranks the list of answers, and presents it to the user. The system obtains five answers after two relaxations. The best three are selected and presented to the user. Two tables are involved: table GEOLOC (50000 tuples) and table RUNWAYS (10 tuples).

```
Elapsed time: 2 seconds processing time for
               relaxation
              10 seconds database retrieval
               time
```

Query 6. Find seaports in Tunisia with a refrigerated storage capacity of over 50 tons.

The relaxation module executes the query. The query is not relaxed, so one database retrieval is performed. Two tables are used: table SEAPORTS (11 tuples) and table GEOLOC (about 50,000 tuples).

```
Elapsed time: 2 seconds processing time for
               relaxation
              5 seconds database retrieval
               time
```

The association module returns relevant information about the seaports. It compares the user query to previous similar cases and selects a set of attributes relevant to the query. Two top-associated attributes are selected and appended to the query. CoBase executes the appended query and returns the answers to the user, together with the additional information. The two additional attributes associated are location name and availability of railroad facility near the seaports.

```
Elapsed time: 10 seconds for association
               computation time
```

Query 7. Find at least 100 cargos of code '3FKAK' with the given volume (length, width, height), code is nonrelaxable.

The relaxation module executes the query and relaxes the height, width, and length according to MTAH, until at least 100 answers are returned. The query is relaxed four times. Thus, five database retrievals are performed. Among the tables accessed is table CARGO_DETAILS (200,000 tuples), a very large table.

```
Elapsed time: 3 seconds processing time for
               relaxation using MTAH
              2 minutes database retrieval time
               for 5 retrievals
```

By using single TAHs (i.e., single TAHs for height, width, and length, respectively), the query is relaxed 12 times. Thus, 13 database retrievals are performed.

```
Elapsed time: 4 seconds for relaxation by
               single TAHs
              5 minutes database retrieval time
               for 13 retrievals
```

For queries involving multiple attributes in the same relation, using an MTAH that covers multiple attributes would provide better relaxation control than using a combination of single-attribute TAHs. The MTAH compares favorably with multiple single-attribute TAHs in both quality and efficiency. We have shown that an MTAH yields a better relaxation strategy than multiple single-attribute TAHs. The primary reason is that MTAHs capture attribute-dependent relationships that cannot be captured when using multiple single-attribute TAHs.

Using MTAHs to control relaxation is more efficient than using multiple single-attribute TAHs. For this example, relaxation using MTAHs require an average of 2.5 relaxation steps, whereas single-attribute TAHs require 8.4 steps. Because a database query is posed after each relaxation step, using MTAHs saves around six database accesses on average. Depending on the size of tables and joins involved, each database access may take from 1 s to about 30 s. As a result, using MTAHs to control relaxation saves a significant amount of user time.

With the aid of domain experts, these queries can be answered by conventional databases. Such an approach takes a few minutes to a few hours. However, without the aid of the domain experts, it may take hours to days to answer these queries. CoBase incorporates domain knowledge as well as relaxation techniques to enlarge the search scope to generate the query answers. Relaxation control plays an important role in enabling the user to control the relaxation process via relaxation control operators such as relaxation order, nonrelaxable attributes, preference list, etc., to restrict the search scope. As a result, CoBase is able to derive the desired answers for the user in significantly less time.

TECHNOLOGY TRANSFER OF COBASE

CoBase stemmed from the transportation planning application for relaxing query conditions. CoBase was linked with SIMS (43) and LIM (44) as a knowledge server for the planning system. SIMS performs query optimizations for distributed databases, and LIM provides high-level language query input to the database. A Technical Integration Experiment was performed to demonstrate the feasibility of this integrated approach. CoBase technology was implemented for the ARPI transportation application (45). Recently, CoBase has also been integrated into a logistical planning tool called Geographical Logistics Anchor Desk (GLAD) developed by GTE/BBN. GLAD is used in locating desired assets for logistical planning which has a very large database (some of the tables exceed one million rows). CoBase has been successfully inserted into GLAD (called CoGLAD), generating the TAHs from the databases, providing similarity search when exact match of the desired assets are not available, and also locating the required amount of these assets with spatial relaxation techniques. The spatial relaxation avoids searching and filtering the entire available assets, which greatly reduces the computation time.

In addition, CoBase has also been successfully applied to the following domains. In electronic warfare, one of the key problems is to identify and locate the emitter for radiated electromagnetic energy based on the operating parameters of observed signals. The signal parameters are radio frequency, pulse repetition frequency, pulse duration, scan period, and the like. In a noisy environment, these parameters often cannot be matched exactly within the emitter specifications. CoBase can be used to provide approximate matching of these emitter signals. A knowledge base (TAH) can be constructed from the parameter values of previously identified signals and also from the peak (typical, unique) parameter values. The TAH provides guidance on the parameter relaxation. The matched emitters from relaxation can be ranked according to relaxation errors. Our preliminary results have shown that CoBase can significantly improve emitter identification as compared to conventional database techniques, particularly in a noisy environment. From the line of bearing of the emitter signal, CoBase can locate the platform that generates the emitter signal by using the near-to relaxation operator.

In medical databases that store X rays and magnetic resonance images, the images are evolution and temporal-based. Furthermore, these images need to be retrieved by object features or contents rather than patient identification (46). The queries asked are often conceptual and not precisely defined. We need to use knowledge about the application (e.g., age class, ethnic class, disease class, bone age), user profile and query context to derive such queries (47). Further, to match the feature exactly is very difficult if not impossible. For example, if the query "Find the treatment methods used for tumors *similar to* X_i ($location_{x_i}$, $size_{x_i}$) on 12-year-old Korean males" cannot be answered, then, based on the TAH shown in Fig. 10, we can relax tumor X_i to tumor Class X, and 12-year-old Korean male to pre-teen Asian, which results in the following relaxed query: "Find the treatment methods used for tumor Class X on pre-teen Asians." Further, we can obtain such relevant information as the success rate, side effects, and cost of the treatment from the association operations. As a result, query relaxation and modification are essential to process these queries. We have applied CoBase technology to medical imaging databases (48). TAHs are generated automatically based on context-specific (e.g., brain tumor) image features (e.g., location, size, shape). After the TAHs for the medical image features have been constructed, query relaxation and modification can be carried out on the medical features (49).

The use of CoSQL constructs such as similar-to, near-to, and within can be used in combination, thus greatly increasing the expressibility for relaxation. For example, we can express "Find tumors *similar-to* the tumor x *based-on* (shape, size, location) and *near-to* object O *within* a specific range (e.g., angle of coverage)." The relaxation control operators, such as matching tumor features in accordance to their importance, can be specified by the operator *relaxation-order* (location, size, shape), to improve the relaxation quality.

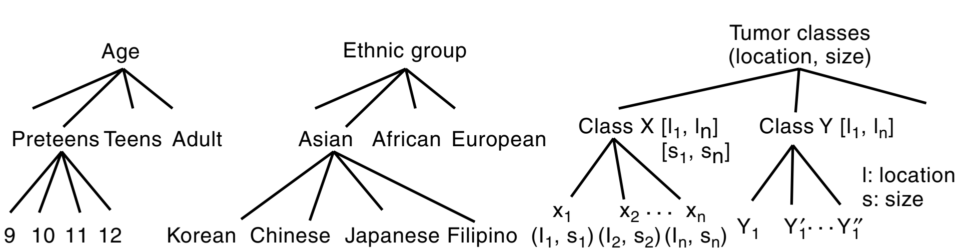

Figure 10. Type abstraction hierarchies for the medical query example.

CONCLUSIONS

After discussing an overview of cooperative database systems, which includes such topics as presuppositions, misconceptions, intensional query answering, user modeling, query relaxation, and associative query answering, we presented a structured approach to query relaxation via Type Abstraction Hierarchy (TAH) and a case-based reasoning approach to provide associative query answering. TAHs are user- and context-sensitive and can be generated automatically from data sources for both numerical and nonnumerical attributes. Therefore, such an approach for query relaxation can scale to large database systems. A set of cooperative operators for relaxation and relaxation control was presented in which these operators were extended to SQL to form a cooperative SQL (CoSQL). A cooperative database (CoBase) has been developed to automatically translate CoSQL queries into SQL queries and can thus run on top of conventional relational databases to provide query relaxation and relaxation control.

The performance measurements on sample queries from CoBase reveal that the cost for relaxation and association is fairly small. The major cost is due to database retrieval which depends on the amount of relaxation required before obtaining a satisfactory answer. The CoBase query relaxation technology has been successfully transferred to the logistics planning application to provide relaxation of asset characteristics as well as spatial relaxation to locate the desired amount of assets. It has also been applied in a medical imaging database (X ray, MRI) for approximate matching of image features and contents, and in electronic warfare for approximate matching of emitter signals (based on a set of parameter values) and also for locating the platforms that generate the signals via spatial relaxation.

With the recent advances in voice recognition systems, more and more systems will be providing voice input features. However, there are many ambiguities in the natural language. Further research in cooperative query answering techniques will be useful in assisting systems to understand users' dialogue with the system.

ACKNOWLEDGMENTS

The research and development of CoBase has been a team effort. I would like to acknowledge the past and present CoBase members—Hua Yang, Gladys Kong, X. Yang, Frank Meng, Guogen Zhang, Wesley Chuang, Meng-feng Tsai, Henrick Yau, and Gilles Fouques—for their contributions toward its design and implementation. The author also wishes to thank the reviewers for their valuable comments.

BIBLIOGRAPHY

1. T. Gaasterland, P. Godfrey, and J. Minker, An overview of cooperative answering, *J. Intell. Inf. Sys.,* **1**: 123–157, 1992.
2. A. Colmerauer and J. Pique, About natural logic, in H. Gallaire, et al. (eds.), *Proc. 5th ECAI,* Orsay, France, 1982, pp. 343–365.
3. S. J. Kaplan, Cooperative responses from a portable natural language query system, *Artificial Intelligence,* **19** (2): 165–187, 1982.
4. E. Mays, Correcting misconceptions about database structure, *Proc. CSCSI 80,* 1980.
5. K. McCoy, Correcting object-related misconceptions, *Proc. COLING10,* Stanford, CA, 1984.
6. L. Cholvy and R. Demolombe, Querying a rule base, *Proc. 1st Int. Conf. Expert Database Syst.,* 1986, pp. 365–371.
7. T. Imielinski, Intelligent Query Answering in Rule Based Systems, in J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming,* Washington, DC: Morgan Kaufman, 1988.
8. A. Motro, Using integrity constraints to provide intensional responses to relational queries, *Proc. 15th Int. Conf. Very Large Data Bases,* Los Altos, CA, 1989, pp. 237–246.
9. A. Pirotte, D. Roelants, and E. Zimanyi, Controlled generation of intensional answers, *IEEE Trans. Knowl. Data Eng.,* **3**: 221–236, 1991.
10. U. Chakravarthy, J. Grant, and J. Minker, Logic based approach to semantic query optimization, *ACM Trans. Database Syst.,* **15** (2): 162–207, 1990.
11. C. Shum and R. Muntz, Implicit Representation for Extensional Answers, in L. Kershberg (ed.), *Expert Database Systems,* Menlo Park, CA: Benjamin/Cummings, 1986, pp. 497–522.
12. W. W. Chu, R. C. Lee, and Q. Chen, Using type inference and induced rules to provide intensional answers, *Proc. IEEE Comput. Soc. 7th Int. Conf. Data Eng.,* Washington, DC, 1991, pp. 396–403.
13. A. Motro, Intensional answers to database queries, *IEEE Trans. Knowl. Database Eng.,* **6** (3): 1994, pp. 444–454.
14. F. Cuppens and R. Demolombe, How to recognize interesting topics to provide cooperative answering, *Inf. Syst.,* **14** (2): 163–173, 1989.
15. W. W. Chu and G. Zhang, Associative query answering via query feature similarity, *Int. Conf. Intell. Inf. Syst.,* pp. 405–501, Grand Bahama Island, Bahamas, 1997.
16. T. Gaasterland, J. Minker, and A. Rajesekar, Deductive database systems and knowledge base systems, *Proc. VIA 90,* Barcelona, Spain, 1990.
17. B. L. Webber and E. Mays, Varieties of user misconceptions: Detection and correction, *Proc. 8th Int. Conf. Artificial Intell.,* Karlsruhe, Germany, 1983, pp. 650–652.
18. W. Wahlster et al., Over-answering yes-no questions: Extended responses in a NL interface to a vision system, *Proc. IJCAI 1983,* Karlsruhe, West Germany, 1983.
19. A. K. Joshi, B. L. Webber, and R. M. Weischedel, Living up to expectations: Computing expert responses, *Proc. Natl. Conf. Artificial Intell.,* Univ. Texas at Austin: The Amer. Assoc. Artif. Intell., 1984, pp. 169–175.
20. J. Allen, *Natural Language Understanding,* Menlo Park, CA: Benjamin/Cummings.
21. S. Carberry, Modeling the user's plans and goals, *Computational Linguistics,* **14** (3): 23–37, 1988.
22. K. F. McCoy, Reasoning on a highlighted user model to respond to misconceptions, *Computational Linguistics,* **14** (3): 52–63, 1988.
23. A. Quilici, M. G. Dyer, and M. Flowers, Recognizing and responding to plan-oriented misconceptions, *Computational Linguistics,* **14** (3): 38–51, 1988.
24. A. S. Hemerly, M. A. Casanova, and A. L. Furtado, Exploiting user models to avoid misconstruals, in R. Demolombe and T. Imielinski (eds.), *Nonstandard Queries and Nonstandard Answers,* Great Britain, Oxford Science, 1994, pp. 73–98.
25. A. Motro, FLEX: A tolerant and cooperative user interface to database, *IEEE Trans. Knowl. Data Eng.,* **4**: 231–246, 1990.
26. W. W. Chu, Q. Chen, and R. C. Lee, Cooperative query answering via type abstraction hierarchy, in S. M. Deen (ed.), *Cooperating Knowledge Based Systems,* Berlin: Springer-Verlag, 1991, pp. 271–292.

27. W. W. Chu and K. Chiang, Abstraction of high level concepts from numerical values in databases, *Proc. AAAI Workshop Knowl. Discovery Databases,* 1994.

28. W. W. Chu et al., An error-based conceptual clustering method for providing approximate query answers [online], *Commun. ACM, Virtual Extension Edition* **39** (12): 216–230, 1996. Available http://www.acm.org/cacm/extension.

29. M. Merzbacher and W. W. Chu, Pattern-based clustering for database attribute values, *Proc. AAAI Workshop on Knowl. Discovery,* Washington, DC, 1993.

30. W. W. Chu and Q. Chen, A structured approach for cooperative query answering, *IEEE Trans. Knowl. Data Eng.,* **6**: 738–749, 1994.

31. W. Chu et al., A scalable and extensible cooperative information system, *J. Intell. Inf. Syst.,* pp. 223–259, 1996.

32. T. Gaasterland, P. Godfrey, and J. Minker, Relaxation as a platform of cooperative answering, *J. Intell. Inf. Syst.,* **1**: 293–321, 1992.

33. G. Fouque, W. W. Chu, and H. Yau, A case-based reasoning approach for associative query answering, *Proc. 8th Int. Symp. Methodologies Intell. Syst.,* Charlotte, NC, 1994.

34. D. H. Fisher, Knowledge acquisition via incremental conceptual clustering, *Machine Learning,* **2** (2): 139–172, 1987.

35. M. A. Gluck and J. E. Corter, Information, uncertainty, and the unity of categories, *Proc. 7th Annu. Conf. Cognitive Sci. Soc.,* Irvine, CA, 1985, pp. 283–287.

36. Y. Cai and N. Cercone, and J. Han, Attribute-Oriented Induction in Relational Databases, in G. Piatetsky-Shapiro and W. J. Frawley (eds.), *Knowledge Discovery in Databases,* Menlo Park, CA: 1991.

37. J. R. Quinlan, The Effect of Noise on Concept Learning, in R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), *Machine Learning,* volume 2, 1986.

38. R. E. Stepp III and R. S. Michalski, Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects, in R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), *Machine Learning,* 1986.

39. T. Finin et al., KQML as an agent communication language, *Proc. 3rd Int. Conf. Inf. Knowl. Manage.,* Gaithersburg, MD, 1994, pp. 456–463.

40. D. M. Mark and A. U. Frank, Concepts of space and spatial language, *Proc. 9th Int. Symp. Comput.-Assisted Cartography,* Baltimore, MD, 1989, pp. 538–556.

41. R. Subramanian and N. R. Adam, Ill-defined spatial operators in geographic databases: Their nature and query processing strategies, *Proc. ACM Workshop Advances Geographical Inf. Syst.,* Washington, DC, 1993, pp. 88–93.

42. A. S. Hemerly, A. L. Furtado, and M. A. Casanova, Towards cooperativeness in geographic databases, *Proc. 4th Int. Conf. Database Expert Syst. Appl.,* Prague, Czech Republic, 1993.

43. Y. Arens and C. Knoblock, Planning and reformulating queries for semantically-modelled multidatabase systems, *Proc. 1st Int. Conf. Inf. Knowl. Manage. (CIKM),* Baltimore, MD, 1992, pp. 92–101.

44. D. P. McKay, J. Pastor, and T. W. Finin, View-concepts: Knowledge-based access to databases, *Proc. 1st Int. Conf. Inf. Knowl. Manage. (CIKM),* Baltimore, MD, 1992, pp. 84–91.

45. J. Stillman and P. Bonissone, Developing new technologies for the ARPA-Rome Planning Initiative, *IEEE Expert,* **10** (1): 10–16, Feb. 1995.

46. W. W. Chu, I. T. Ieong, and R. K. Taira, A semantic modeling approach for image retrieval by content, *J. Very Large Database Syst.,* **3**: 445–477, 1994.

47. W. W. Chu, A. F. Cardenas, and R. K. Taira, KMeD: A knowledge-based multimedia medical distributed database system, *Inf. Syst.,* **20** (2): 75–96, 1995.

48. H. K. Huang and R. K. Taira, Infrastructure design of a picture archiving and communication system, *Amer. J. Roentgenol.,* **158**: 743–749, 1992.

49. C. Hsu, W. W. Chu, and R. K. Taira, A knowledge-based approach for retrieving images by content, *IEEE Trans. Knowl. Data Eng.,* **8**: 522–532, 1996.

WESLEY W. CHU
University of California
at Los Angeles

COORDINATE TRANSFORMATIONS, OPTICAL.

See OPTICAL AND ELECTRO-OPTICAL IMAGE CONVERTERS.

COPPER-INDIUM-DISELENIDE BASED SOLAR CELLS.

See TERNARY SEMICONDUCTORS.

CORE LOSSES.

See EDDY CURRENT LOSSES.

CORNER DETECTION.

See FEATURE EXTRACTION.