Franz Rothlauf

Representations for Genetic and Evolutionary Algorithms

Second Edition

Springer

Representations for Genetic and Evolutionary Algorithms

Franz Rothlauf

Representations for Genetic and Evolutionary Algorithms

Springer

Dr. Franz Rothlauf
Universität Mannheim
68131 Mannheim
Germany
E-mail: rothlauf@uni-mannheim.de

Library of Congress Control Number: 2005936356

ISBN-10 3-540-25059-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-25059-3 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media
springer.com

Printed in The Netherlands

Typesetting: by the author and TechBooks using a Springer LaTeX macro package

Cover design: *Erich Kirchner*, Heidelberg

Printed on acid-free paper SPIN: 11370550 89/TechBooks 5 4 3 2 1 0

Für meine Eltern Elisabeth und Alfons Rothlauf.

Preface

Preface to the Second Edition

I have been very pleased to see how well the first edition of this book has been accepted and used by its readers. I have received fantastic feedback telling me that people use it as an inspiration for their own work, give it to colleagues or students, or use it for preparing lectures and classes about representations. I want to thank you all for using the material presented in this book and for developing more efficient and powerful heuristic optimization methods.

You will find this second edition of the book completely revised and extended. The goal of the revisions and extensions was to make it easier for the reader to understand the main points and to get a more thorough knowledge of the design of high-quality representations. For example, I want to draw your attention to Chap. 3 where you find the core of the book. I have extended and improved the sections about redundancy and locality of representations adding new material and experiments and trying to draw a more comprehensive picture. In particular, the introduction of synonymity for redundant encodings in Sect. 3.1 and the integration of locality and redundancy issues in Sect. 3.3 are worth having a closer look at it. These new concepts have been used throughout the work and have made it possible to better understand a variety of different representation issues.

The chapters about tree representations have been reorganized such that they explicitly distinguish between direct and indirect representations. This distinction – including a new analysis of the edge-sets, which is a direct encoding for trees – emphasizes that the developed representation framework is not only helpful for analysis and design of representations, but also for operators. The design of proper search operators is at the core of direct representations and the new sections demonstrate how to analyze the influence of such encodings on the performance of genetic and evolutionary algorithms (GEAs). Finally, the experiments presented in Chap. 8 have been completely revised considering new representations and giving a better understanding of the influence of tree representations on the performance of GEAs.

I would like to take this opportunity to thank everyone who took the time to share their thoughts on the text with me – all these comments were helpful in improving the book. Special thanks to Kati for her support in preparing this work.

As with the first edition, my purpose will be fulfilled if you find this book helpful for building more efficient heuristic optimization methods, if you find it inspiring for your research, or if it is a help for you teaching students about the importance and influence of representations.

Mannheim
August 2005 *Franz Rothlauf*

Preface to the First Edition

This book is about representations for genetic and evolutionary algorithms (GEAs). In writing it, I have tried to demonstrate the important role of representations for an efficient use of genetics and evolutionary optimization methods. Although, experience often shows that the choice of a proper representation is crucial for GEA's success, there are few theoretical models that describe how representations influence GEAs behavior. This book aims to resolve this unsettled situation. It presents theoretical models describing the effect of different types of representations and applies them to binary representations of integers and tree representations.

The book is designed for people who want to learn some theory about how representations influence GEA performance and for those who want to see how this theory can be applied to representations in the real world. The book is based on my dissertation with the title "Towards a Theory of Representations for Genetic and Evolutionary Algorithms: Development of Basic Concepts and their Application to Binary and Tree Representations". To make the book easier to read for a larger audience some chapters are extended and many explanations are more detailed. During the writing of the book many people from various backgrounds (economics, computer science and engineering) had a look at the work and pushed me to present it in a way that is accessible to a diverse audience. Therefore, also people that are not familiar to GEAs should be able to get the basic ideas of the book.

To understand the theoretical models describing the influence of representations on GEA performance I expect college-level mathematics like elementary notions of counting, probability theory and algebra. I tried to minimize the mathematical background required for understanding the core lessons of the book and to give detailed explanations on complex theoretical subjects. Furthermore, I expect the reader to have no particular knowledge of genetics and define all genetic terminology and concepts in the text. The influence of

integer and tree representations on GEA performance does not necessarily require a complete understanding of the elements of representation theory but is also accessible for people who do not want to bother too much with theory.

The book is split up into two large parts. The first presents theoretical models describing the effects of representations on GEA performance. The second part uses the theory for the analysis and design of representations. After the first two introductory chapters, theoretical models are presented on how redundant representations, exponentially scaled representations and the locality/distance distortion of a representation influence GEA performance. In Chap. 4 the theory is used for formulating a time-quality framework. Consequently, in Chap. 5, the theoretical models are used for analyzing the performance differences between binary representations of integers. Finally, the framework is used in Chap. 6, Chap. 7, and Chap. 8 for the analysis of existing tree representations as well as the design of new tree representations. In the appendix common test instances for the optimal communication spanning tree problems are summarized.

Acknowledgments

First of all, I would like to thank my parents for always providing me with a comfortable home environment. I have learned to love the wonders of the world and what the important things in life are.

Furthermore, I would like to say many thanks to my two advisors, Dr. Armin Heinzl and Dr. David E. Goldberg. They did not only help me a lot with my work, but also had a large impact on my private life. Dr. Armin Heinzl helped me to manage my life in Bayreuth and always guided me in the right direction in my research. He was a great teacher and I was able to learn many important things from him. I am grateful to him for creating an environment that allowed me to write this book. Dr. David E. Goldberg had a large influence on my research life. He taught me many things which I needed in my research and I would never have been able to write this thesis without his help and guidance.

During my time here in Bayreuth, my colleagues in the department have always been a great help to overcome the troubles of daily university life. I especially want to thank Michael Zapf, Lars Brehm, Jens Dibbern, Monika Fortmühler, Torsten O. Paulussen, Jürgen Gerstacker, Axel Pürckhauer, Thomas Schoberth, Stefan Hocke, and Frederik Loos. During my time here, Wolfgang Güttler and Tobias Grosche were not only work colleagues, but also good friends. I want to thank them for the good time I had and the interesting discussions.

During the last three years during which I spent time at IlliGAL I met many people who have had a great impact on my life. First of all, I would like to thank David E. Goldberg and the Department of General Engineering for giving me the opportunity to stay there so often. Then, I want to say thank you

to the folks at IlliGAL I was able to work together with. It was always a really great pleasure. I especially want to thank Erick Cantú-Paz, Fernando Lobo, Dimitri Knjazew, Clarissa van Hoyweghen, Martin Butz, Martin Pelikan, and Kumara Sastry. It was not only a pleasure working together with them but over time they have become really good friends. My stays at IlliGAL would not have been possible without their help.

Finally, I want to thank the people who were involved in the writing of this book. First of all I want to thank Kumara Sastry and Martin Pelikan again. They helped me a lot and had a large impact on my work. The discussions with Martin were great and Kumara often impressed me with his expansive knowledge about GEAs. Then, I want to say thanks to Fernando Lobo and Torsten O. Paulussen. They gave me great feedback and helped me to clarify my thoughts. Furthermore, Katrin Appel and Kati Sternberg were a great help in writing this dissertation. Last but not least I want to thank Anna Wolf. Anna is a great proofreader and I would not have been able to write a book in readable English without her help.

Finally, I want to say "thank you" to Kati. Now I will hopefully have more time for you.

Bayreuth
January 2002 *Franz Rothlauf*

Foreword to the First Edition

It is both personally and intellectually pleasing for me to write a foreword to this work. In January 1999 I received a brief e-mail from a PhD student at the University of Bayreuth asking if he might visit the Illinois Genetic Algorithms Laboratory (IlliGAL). I did not know this student, Franz Rothlauf, but something about the tone of his note suggested a sharp, eager mind connected to a cheerful, positive nature. I checked out Franz's references, invited him to Illinois for a first visit, and my early feelings were soon proven correct. Franz's various visits to the lab brought both smiles to the faces of IlliGAL labbies and important progress to a critical area of genetic algorithm inquiry. It was great fun to work with Franz and it was exciting to watch this work take shape. In the remainder, I briefly highlight the contributions of this work to our state of knowledge.

In the field of genetic and evolutionary algorithms (GEAs), much theory and empirical study has been heaped upon operators and test problems, but problem representation has often been taken as a given. In this book, Franz breaks with this tradition and seriously studies a number of critical elements of a theory of GEA representation and applies them to the careful empirical study of (a) a number of important idealized test functions and (b) problems of commercial import. Not only is Franz creative in *what* he has chosen to study, he also has been innovative in *how* he performs his work.

In GEAs – as elsewhere – there appears sometimes to be a firewall separating theory and practice. This is not new, and even Bacon commented on this phenomenon with his famous metaphor of the spiders (men of dogmas), the ants (men of practice), and the bees (transformers of theory to practice). In this work, Franz is one of Bacon's bees, taking *applicable* theory of representation and carrying it to practice in a manner that (1) illuminates the theory and (2) answers the questions of importance to a practitioner.

This book is original in many respects, so it is difficult to single out any one of its many accomplishments. I do believe five items deserve particular comment:

1. Decomposition of the representation problem
2. Analysis of redundancy

3. Analysis of scaling
4. Time-quality framework for representation
5. Demonstration of the framework in well-chosen test problems and problems of commercial import.

Franz's decomposition of the problem of representation into issues of redundancy, scaling, and correlation is itself a contribution. Individuals have isolated each of these areas previously, but this book is the first to suggest they are core elements of an integrated theory and to show the way toward that integration.

The analyses of redundancy and scaling are examples of *applicable* or *facetwise* modeling at its best. Franz gets at key issues in run duration and population size through bounding analyses, and these permit him to draw definite conclusions in fields where so many other researchers have simply waived their arms.

By themselves, these analyses would be sufficient, but Franz then takes the extra and unprecedented step toward an integrated *quality-time* framework for representations. The importance of quality and time has been recognized previously from the standpoint of operator design, but this work is the first to understand that codings can and should be examined from an efficiency-quality standpoint as well. In my view, this recognition will be understood in the future as a key turning point away from the current voodoo and black magic of GEA representation toward a scientific discussion of the appropriateness of particular representations for different problems.

Finally, Franz has carefully demonstrated his ideas in (1) carefully chosen test functions and (2) problems of commercial import. Too often in the GEA field, researchers perform an exercise in pristine theory without relating it to practice. On the other hand, practitioners too often study the latest wrinkle in problem representation or coding without theoretical backing or support. This dissertation asserts the applicability of its theory by demonstrating its utility in understanding tree representations, both test functions and real-world communications networks. Going from theory to practice in such a sweeping manner is a rare event, and the accomplishment must be regarded as both a difficult and an important one.

All this would be enough for me to recommend this book to GEA aficionados around the globe, but I hasten to add that the book is also remarkably well written and well organized. No doubt this rhetorical craftsmanship will help broaden the appeal of the book beyond the ken of genetic algorithmists and computational evolutionaries. In short, I recommend this important book to anyone interested in a better quantitative and qualitative understanding of the representation problem. Buy this book, read it, and use its important methodological, theoretical, and practical lessons on a daily basis.

University of Illinois at Urbana-Champaign *David E. Goldberg*

Contents

1

Introduction

One of the major challenges for researchers in the field of management science, information systems, business informatics, and computer science is to develop methods and tools that help organizations, such as companies or public institutions, to fulfill their tasks efficiently. However, during the last decade, the dynamics and size of tasks organizations are faced with has changed. Firstly, production and service processes must be reorganized in shorter time intervals and adapted dynamically to the varying demands of markets and customers. Although there is continuous change, organizations must ensure that the efficiency of their processes remains high. Therefore, optimization techniques are necessary that help organizations to reorganize themselves, to increase the performance of their processes, and to stay efficient. Secondly, with increasing organization size the complexity of problems in the context of production or service processes also increases. As a result, standard, traditional, optimization techniques are often not able to solve these problems of increased complexity with justifiable effort in an acceptable time period. Therefore, to overcome these problems, and to develop systems that solve these complex problems, researchers proposed using genetic and evolutionary algorithms (GEAs). Using these nature-inspired search methods it is possible to overcome some limitations of traditional optimization methods, and to increase the number of solvable problems. The application of GEAs to many optimization problems in organizations often results in good performance and high quality solutions.

For successful and efficient use of GEAs, it is not enough to simply apply standard GEAs. In addition, it is necessary to find a proper representation for the problem and to develop appropriate search operators that fit well to the properties of the representation. The representation must at least be able to encode all possible solutions of an optimization problem, and genetic operators such as crossover and mutation should be applicable to it.

Many optimization problems can be encoded by a variety of different representations. In addition to traditional binary and continuous string encodings, a large number of other, often problem-specific representations have been

proposed over the last few years. Unfortunately, practitioners often report a significantly different performance of GEAs by simply changing the used representation. These observations were confirmed by empirical and theoretical investigations. The difficulty of a specific problem, and with it the performance of GEAs, can be changed dramatically by using different types of representations. Although it is well known that representations affect the performance of GEAs, no theoretical models exist which describe the effect of representations on the performance of GEAs. Therefore, the design of proper representations for a specific problem mainly depends on the intuition of the GEA designer and developing new representations is often a result of repeated trial and error. As no theory of representations exists, the current design of proper representations is not based on theory, but more a result of black art.

The lack of existing theory not only hinders a theory-guided design of new representations, but also results in problems when deciding which of the different representations should be used for a specific optimization problem. Currently, comparisons between representations are based mainly on limited empirical evidence, and random or problem-specific test function selection. However, empirical investigations only allow us to judge the performance of representations for the specific test problem, but do not help us in understanding the basic principles behind it. A representation can perform well for many different test functions, but fails for the one problem which one really wants to solve. If it is possible to develop theoretical models which describe the influence of representations on measurements of GEA performance – like time to convergence and solution quality – then representations can be used efficiently and in a theory-guided manner. Choosing and designing proper representations will not remain the black art of GEA research but become a well predictable engineering task.

1.1 Purpose

The purpose of this work is to bring some order into the unsettled situation which exists and to investigate how representations influence the performance of genetic and evolutionary algorithms. This work develops elements of representation theory and applies them to designing, selecting, using, choosing among, and comparing representations. It is not the purpose of this work to substitute the current black art of choosing representations by developing barely applicable, abstract, theoretical models, but to formulate an applicable representation theory that can help researchers and practitioners to find or design the proper representation for their problem. By providing an applicable theory of representations this work should bring us to a point where the influence of representations on the performance of GEAs can be judged easily and quickly in a theory-guided manner.

The first step in the development of an applicable theory is to identify which properties of representations influence the performance of GEAs and

how. Therefore, this work models for different properties of representations how solution quality and time to convergence is changed. Using this theory, it is possible to formulate a framework for efficient design of representations. The framework describes how the performance of GEAs, measured by run duration and solution quality, is affected by the properties of a representation. By using this framework, the influence of different representations on the performance of GEAs can be explained. Furthermore, it allows us to compare representations in a theory-based manner, to predict the performance of GEAs using different representations, and to analyze and design representations guided by theory. One does not have to rely on empirical studies to judge the performance of a representation for a specific problem, but can use existing theory for predicting GEA performance. By using this theory, the situation exists where empirical results are only needed to validate theoretical predictions.

However, developing a general theory of how representations affect GEA performance is a demanding and difficult task. To simplify the problem, it must be decomposed, and the different properties of encodings must be investigated separately. Three different properties of representations are considered in this work: Redundancy, scaling, and locality, respectively distance distortion. For these three properties of representations models are developed that describe their influence on the performance of GEAs. Additionally, population sizing and time to convergence models are presented for redundant and non-uniformly scaled encodings. Furthermore, it is shown that low-locality representations can change the difficulty of the problem. For low-locality encodings, it can not exactly be predicted how GEA performance is changed, without having complete knowledge regarding the structure of the problem. Although the investigation is limited only to three important properties of representations, the understanding of the influence of these three properties of encodings on the performance of GEAs brings us a large step forward towards a general theory of representations.

To illustrate the significance and importance of the presented representation framework on the performance of GEAs, the framework is used for analyzing the performance of binary representations of integers and tree representations. The investigations show that the current framework considering only three representation properties gives us a good understanding of the influence of representations on GEA performance as it allows us to predict the performance of GEAs using different types of representations. The results confirm that choosing a proper representation has a large impact on the performance of GEAs, and therefore, a better theoretical understanding of representations is necessary for an efficient use of genetic search.

Finally, it is illustrated how the presented theory of representations can help us in designing new representations more reasonably. It is shown by example for tree representations, that the presented framework allows theory-guided design. Not black art, but a deeper understanding of representations allows us to develop representations which result in a high performance of genetic and evolutionary algorithms.

1.2 Organization

The organization of this work follows its purpose. It is divided into two large parts: After the first two introductory chapters, the first part (Chaps. 3 and 4) provides the theory regarding representations. The second part (Chaps. 5, 6, 7, and 8) applies the theory to the analysis and design of representations. Chapter 3 presents theory on how different properties of representations affect GEA performance. Consequently, Chap. 4 uses the theory for formulating the time-quality framework. Then, in Chap. 5, the presented theory of representations is used for analyzing the performance differences between binary representations of integers. Finally, the framework is used in Chap. 6, Chap. 7, and Chap. 8 for the analysis and design of tree representations and search operators. The following paragraphs give a more detailed overview about the contents of each chapter.

Chapter 1 is the current chapter. It sets the stage for the work and describes the benefits that can be gained from a deeper understanding of representations for GEAs.

Chapter 2 provides the background necessary for understanding the main issues of this work about representations for GEAs. Section 2.1 introduces representations which can be described as a mapping that assigns one or more genotypes to every phenotype. The genetic operators selection, crossover, and mutation are applied on the level of alleles to the genotypes, whereas the fitness of individuals is calculated from the corresponding phenotypes. Section 2.2 illustrates that selectorecombinative GEAs, where only crossover and selection operators are used, are based on the notion of schemata and building blocks. Using schemata and building blocks is an approach to explain why and how GEAs work. This is followed in Sect. 2.3 by a brief review of reasons and measurements for problem difficulty. Measurements of problem difficulty are necessary to be able to compare the influence of different types of representations on the performance of GEAs. The chapter ends with some earlier, mostly qualitative recommendations for the design of efficient representations.

Chapter 3 presents three aspects of a theory of representations for GEAs. It investigates how redundant encodings, encodings with exponentially scaled alleles, and representations that modify the distances between the corresponding genotypes and phenotypes, influence GEA performance. Population sizing models and time to convergence models are presented for redundant and exponentially scaled representations. Section 3.1 illustrates that redundant encodings influence the supply of building blocks in the initial population of GEAs. Based on this observation the population sizing model from Harik et al. (1997) and the time to convergence model from Thierens and Goldberg (1993) can be extended from non-redundant to redundant representations. Because redundancy mainly affects the number of copies in the initial population that are given to the optimal solution, redundant representations increase solution quality and reduce time to convergence if individuals that are similar to the optimal solution are overrepresented. Section 3.2 focuses on exponen-

tially scaled representations. The investigation into the effects of exponentially scaled encodings shows that, in contrast to uniformly scaled representations, the dynamics of genetic search are changed. By combining the results from Harik et al. (1997) and Thierens (1995) a population sizing model for exponentially scaled building blocks with and without considering genetic drift can be presented. Furthermore, the time to convergence when using exponentially scaled representations is calculated. The results show that when using non-uniformly scaled representations, the time to convergence increases. Finally, Sect. 3.3 investigates the influence of representations that modify the distances between corresponding genotypes and phenotypes on the performance of GEAs. When assigning the genotypes to the phenotypes, representations can change the distances between the individuals. This effect is denoted as locality or distance distortion. Investigating its influence shows that the size and length of the building blocks, and therefore the complexity of the problem are changed if the distances between the individuals are not preserved. Therefore, to ensure that an easy problem remains easy, high-locality representations which preserve the distances between the individuals are necessary.

Chapter 4 presents the framework for theory-guided analysis and design of representations. The chapter combines the three elements of representation theory from Chap. 3 – redundancy, scaling, and locality – to a time-quality framework. It formally describes how the time to convergence and the solution quality of GEAs depend on these three aspects of representations. The chapter ends with implications for the design of representations which can be derived from the framework. In particular, the framework tells us that uniformly scaled representations are robust, that exponentially scaled representations are fast but inaccurate, and that low-locality representations change the difficulty of the underlying optimization problem.

Chapter 5 uses the framework for a theory-guided analysis of binary representations of integers. Because the potential number of schemata is higher when using binary instead of integer representations, users often favor the use of binary instead of integer representations, when applying GEAs to integer problems. By using the framework it can be shown that the redundant unary encoding results in low GEA performance if the optimal solution is underrepresented. Both, Gray and binary encoding are low-locality representations as they change the distances between the individuals. Therefore, both representations change the complexity of optimization problems. It can be seen that the easy integer one-max problem is easier to solve when using the binary representation, and the difficult integer deceptive trap is easier to solve when using the Gray encoding.

Chapter 6 uses the framework for the analysis and design of tree representations. For tree representations, standard crossover and mutation operators are applied to tree-specific genotypes. However, finding or defining tree-specific genotypes and genotype-phenotype mappings is a difficult task because there are no intuitive genotypes for trees. Therefore, researchers have proposed a variety of different, more or less tricky representations which can be used in

combination with standard crossover and mutation operators. A closer look at the Prüfer number representation in Sect. 6.2 reveals that the encoding in general is a low-locality representation and modifies the distances between corresponding genotypes and phenotypes. As a result, problem complexity is modified, and many easy problems become too difficult to be properly solved using GEAs. Section 6.3 presents an investigation into the characteristic vector representation. Because invalid solutions are possible when using characteristic vectors, an additional repair mechanism is necessary which makes the representation redundant. Characteristic vectors are uniformly redundant and GEA performance is independent of the structure of the optimal solution. However, the repair mechanism results in non-synonymous redundancy. Therefore, GEA performance is reduced and the time to convergence increases. With increasing problem size, the repair process generates more and more links randomly and offspring trees have not much in common with their parents. Therefore, for larger problems guided search is no longer possible and GEAs behave like random search. In Sect. 6.4, the investigation into the redundant link and node biased representation reveals that the representation overrepresents trees that are either star-like or minimum spanning tree-like. Therefore, GEAs using this type of representation perform very well if the optimal solution is similar to stars or to the minimum spanning tree, whereas they fail when searching for optimal solutions that do not have much in common with stars or the minimum spanning tree. Finally, Sect. 6.5 presents network random keys (NetKeys) as an example for the theory-guided design of a tree representation. To construct a robust and predictable tree representation, it should be non- or uniformly redundant, uniformly scaled, and have high-locality. When combining the concepts of the characteristic vector representation with weighted representations like the link and node biased representation, the NetKey representation can be created. In analogy to random keys, the links of a tree are represented as floating numbers, and a construction algorithm constructs the corresponding tree from the keys. The NetKey representation allows us to distinguish between important and unimportant links, is uniformly redundant, uniformly scaled, and has high locality.

Chapter 7 uses the insights into representation theory for the analysis and design of search operators for trees. In contrast to Chap. 6 where standard search operators are applied to tree-specific genotypes, now tree-specific search operators are directly applied to tree structures. Such types of representations are also known as direct representations as there is no additional genotype-phenotype mapping. Section 7.1 presents a direct representation for trees (NetDir) as an example for the design of direct tree representations. Search operators are directly applied to trees and problem-specific crossover and mutation operators are developed. The search operators for the NetDir representation are developed based on the notion of schemata. Section 7.2 analyzes the edge-set encoding which encodes trees directly by listing their edges. Search operators for edge-sets may be heuristic, considering the weights of edges they include in offspring, or naive, including edges without

regard to their weights. Analyzing the properties of the heuristic variants of the search operators shows that solutions similar to the minimum spanning tree are favored. In contrast, the naive variants are unbiased which means that genetic search is independent of the structure of the optimal solution. Although no explicit genotype-phenotype mapping exists for edge-sets and the framework for the design of representations cannot be directly applied, the framework is useful for structuring the analysis of edge-sets. Similarly to non-uniformly redundant representations, edge-sets overrepresent some specific types of tree and GEA performance increases if optimal solutions are similar to the MST. Analyzing and developing direct representations nicely illustrates the trade-off between designing either problem-specific representations or problem-specific operators. For efficient GEAs, it is necessary either to design problem-specific representations and to use standard operators like one-point or uniform crossover, or to develop problem-specific operators and to use direct representations.

Chapter 8 verifies theoretical predictions concerning GEA performance by empirical verification. It compares the performance of GEAs using different types of representations for the one-max tree problem, the deceptive tree problem, and various instances of the optimal communication spanning tree problem. The instances of the optimal communication spanning trees are presented in the literature (Palmer 1994; Berry et al. 1997; Raidl 2001; Rothlauf et al. 2002). The results show that with the help of the framework the performance of GEAs using different types of representations can be well predicted.

Chapter 9 summarizes the major contributions of this work, describes how the knowledge about representations has changed, and gives some suggestions for future research.

2

Representations for Genetic and Evolutionary Algorithms

In this second chapter, we present an introduction into the field of representations for genetic and evolutionary algorithms. The chapter provides the basis and definitions which are essential for understanding the content of this work.

Genetic and evolutionary algorithms (GEAs) are nature-inspired optimization methods that can be advantageously used for many optimization problems. GEAs imitate basic principles of life and apply genetic operators like mutation, crossover, or selection to a sequence of alleles. The sequence of alleles is the equivalent of a chromosome in nature and is constructed by a representation which assigns a string of symbols to every possible solution of the optimization problem. Earlier work (Goldberg 1989c; Liepins and Vose 1990) has shown that the behavior and performance of GEAs is strongly influenced by the representation used. As a result, many recommendations for a proper design of representations were made over the last few years (Goldberg 1989c; Radcliffe 1991a; Radcliffe 1991b; Palmer 1994; Ronald 1997). However, most of these design rules are of a qualitative nature and are not particularly helpful for estimating exactly how different types of representations influence problem difficulty. Consequently, we are in need of a theory of representations which allows us to theoretically predict how different types of representations influence GEA performance. This chapter provides some of the utilities that are necessary for reaching this goal.

The chapter starts with an introduction into genetic representations. We describe the notion of genotypes and phenotypes and illustrate how the fitness function can be decomposed into a genotype-phenotype, and a phenotype-fitness mapping. The section ends with a brief characterization of widely used representations. In Sect. 2.2, we provide the basis for genetic and evolutionary algorithms. After a brief description of the principles of a simple genetic algorithm (GA), we present the underlying theory which explains why and how selectorecombinative GAs using crossover as a main search operator work. The schema theorem tells us that GAs process schemata and the building block hypothesis assumes that many real-world problems are decomposable (or at least quasi-decomposable). Therefore, GAs perform well for these types

of problems. Section 2.3 addresses the difficulty of problems. After illustrating that the reasons for problem difficulty depend on the used optimization method, we describe some common measurements of problem complexity. Finally, in Sect. 2.4 we review some former recommendations for the design of efficient representations.

2.1 Genetic Representations

This section introduces representations for genetic and evolutionary algorithms. When using GEAs for optimization purposes, representations are required for encoding potential solutions. Without representations, no use of GEAs is possible.

In Sect 2.1.1, we introduce the notion of genotype and phenotype. We briefly describe how nature creates a phenotype from the corresponding genotype by the use of representations. This more biology-based approach to representations is followed in Sect. 2.1.2 by a more formal description of representations. Every fitness function f which assigns a fitness value to a genotype x^g can be decomposed into the genotype-phenotype mapping f_g, and the phenotype-fitness mapping f_p. Finally, in Sect. 2.1.3 we briefly review the most important types of representations.

2.1.1 Genotypes and Phenotypes

In 1866, Mendel recognized that nature stores the complete genetic information for an individual in pairwise alleles (Mendel 1866). The genetic information that determines the properties, appearance, and shape of an individual is stored by a number of strings. Later, it was discovered that the genetic information is formed by a double string of four nucleotides, called DNA.

Mendel realized that nature distinguishes between the genetic code of an individual and its outward appearance. The genotype represents all the information stored in the chromosomes and allows us to describe an individual on the level of genes. The phenotype describes the outward appearance of an individual. A transformation exists – a genotype-phenotype mapping or a representation – that uses the genotypic information to construct the phenotype. To represent the large number of possible phenotypes with only four nucleotides, the genotypic information is not stored in the alleles itself, but in the sequence of alleles. By interpreting the sequence of alleles, nature can encode a large number of different phenotypic expressions using only a few different types of alleles.

In Fig. 2.1, we illustrate the differences between chromosome, gene, and allele. A chromosome describes a string of certain length where all the genetic information of an individual is stored. Although nature often uses more than one chromosome, most GEA applications only use one chromosome for encoding the genotypic information. Each chromosome consist of many alleles.

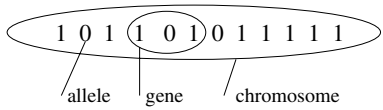


Figure 2.1. Alleles, genes, and chromosomes

Alleles are the smallest information units in a chromosome. In nature, alleles exist pairwise, whereas in most GEA implementations an allele is represented by only one symbol. If for example, we use a binary representation, an allele can have either the value 0 or 1. If a phenotypic property of an individual, like its hair color or eye size is determined by one or more alleles, then these alleles together are denoted to be a gene. A gene is a region on a chromosome that must be interpreted together and which is responsible for a specific phenotypic property.

When talking about individuals in a population, we must carefully distinguish between genotypes and phenotypes. The phenotypic appearance of an individual determines its success in life. Therefore, when comparing the abilities of different individuals we must judge them on the level of the phenotype. However, when it comes to reproduction we must view individuals on the level of the genotype. During sexual reproduction, the offspring does not inherit the phenotypic properties of its parents, but only the genotypic information regarding the phenotypic properties. The offspring inherits genetic material from both parents. Therefore, genetic operators work on the level of the genotype, whereas the evaluation of the individuals is performed on the level of the phenotype.

2.1.2 Decomposition of the Fitness Function

The following subsection provides some basic definitions for our discussion of representations for genetic and evolutionary algorithms. We show how every optimization problem that should be solved by using GEAs can be decomposed into a genotype-phenotype f_g, and a phenotype-fitness mapping f_p.

We define Φ_g as the genotypic search space where the genetic operators such as recombination or mutation are applied to. An optimization problem on Φ_g could be formulated as follows: The search space Φ_g is either discrete or continuous, and the function

$$f(\boldsymbol{x}) : \Phi_g \rightarrow \mathbb{R}$$

assigns an element in $\mathbb{R}$ to every element in the genotype space Φ_g. The optimization problem is defined by finding the optimal solution

$$\hat{\boldsymbol{x}} = \max_{\boldsymbol{x} \in \Phi_g} f(\boldsymbol{x}),$$

where $\boldsymbol{x}$ is a vector of decision variables (or alleles), and $f(\boldsymbol{x})$ is the objective or fitness function. The vector $\hat{\boldsymbol{x}}$ is the global maximum. We have chosen to

illustrate a maximization problem, but without loss of generality, we could also model a minimization problem. To be able to apply GEAs to a problem, the inverse function f^{-1} does not need to exist.

In general, the cardinality of Φ_g can be greater than two, but we want to focus for the most part in our investigation on binary search spaces with cardinality two. Thus, GEAs search in the binary space

$$\Phi_g = \{0,1\}^l,$$

with the length of the string $\boldsymbol{x}$ equal to l and the size of the search space $|\Phi_g| = 2^l$.

The introduction of an explicit representation is necessary if the phenotype of a problem can not be depicted as a string or in another way that is accessible for GEAs. Furthermore, the introduction of a representation could be useful if there are constraints or restrictions on the search space that can be advantageously modeled by a specific encoding. Finally, using the same genotypes for different types of problems, and only interpreting them differently by using a different genotype-phenotype mapping, allows us to use standard genetic operators with known properties. Once we have gained some knowledge about specific kinds of genotypes, we can easily reuse that knowledge, and it is not necessary to develop any new operators.

When using a representation for genetic and evolutionary algorithms we have to introduce – in analogy to nature – phenotypes and genotypes (Lewontin 1974; Liepins and Vose 1990). The fitness function f is decomposed into two parts. The first maps the genotypic space Φ_g to the phenotypic space Φ_p, and the second maps Φ_p to the fitness space $\mathbb{R}$. Using the phenotypic space Φ_p we get:

$$f_g(x^g) : \Phi_g \rightarrow \Phi_p,$$
$$f_p(x^p) : \Phi_p \rightarrow \mathbb{R},$$

where $f = f_p \circ f_g = f_p(f_g(x^g))$. The genotype-phenotype mapping f_g is determined by the type of representation used. f_p represents the fitness function and assigns a fitness value $f_p(x^p)$ to every individual $x^p \in \Phi_p$. The genetic operators are applied to the individuals in Φ_g (Bagley 1967; Vose 1993).

If the genetic operators are applied directly to the phenotype it is not necessary to specify a representation and the phenotype is the same as the genotype:

$$f_g(x^g) : \Phi_g \rightarrow \Phi_g,$$
$$f_p(x^p) : \Phi_g \rightarrow \mathbb{R}.$$

In this case, f_g is the identity function $f_g(x^g) = x^g$. All genotypic properties are transformed to the phenotypic space. The genotypic space is the same as the phenotypic space and we have a *direct representation*. Because there is no longer an additional mapping between Φ_g and Φ_p, a direct representation

does not change any aspect of the phenotypic problem such as complexity, distances between the individuals, or location of the optimal solution. However, when using direct representations, we could not in general use standard genetic operators, but have to define problem-specific operators (see Sect. 7 regarding the analysis of direct representations for trees). Therefore, the key factor for the success of a GEA using a direct representation is not in finding a "good" representation for a specific problem, but in developing proper search operators.

We have seen how every optimization problem we want to solve with GEAs can be decomposed into a genotype-phenotype f_g, and a phenotype-fitness mapping f_p. The genetic operators are applied to the genotypes $x^g \in \Phi_g$, and the fitness of the individuals is calculated from the phenotypes $x^p \in \Phi_p$.

2.1.3 Types of Representations

In this subsection, we describe some of the most important and widely used representations, and summarize some of their major characteristics. In this work, we do not provide an overview of all representations which appear in the literature because every time a GEA is used, some kind of representation is necessary. This means within the scope of this research it is not possible to review all representations which have once been presented. For a more detailed overview about different types of representations see Bäck et al. (1997, Sect. C1).

Binary Representations

Binary representations are the most common representations for selectorecombinative GEAs. Selectorecombinative GEAs process schemata and use crossover as the main search operator. Using these types of GEAs, mutation only serves as background noise. The search space Φ_g is denoted by $\Phi_g = \{0,1\}^l$, where l is the length of a binary vector $x^g = (x_1^g, \ldots x_l^g) \in \{0,1\}^l$ (Goldberg 1989c).

When using binary representations the genotype-phenotype mapping f_g depends on the specific optimization problem that should be solved. For many combinatorial optimization problems the binary representation allows a direct and very natural encoding.

When encoding integer problems by using binary representations, specific genotype-phenotype mappings are necessary. Different types of binary representations for integers assign the integers $x^p \in \Phi_p$ (phenotypes) in a different way to the binary vectors $x^g \in \Phi_g$ (genotypes). The most common representations are the binary, Gray, and unary encoding. For a more detailed description of these three types of encodings see Sect. 5.2.

When encoding continuous variables by using binary vectors the accuracy of the representation depends on the number of bits that represent a phenotypic continuous variable. By increasing the number of bits that are used for

representing one continuous variable the accuracy of the representation can be increased. When encoding a continuous phenotypic variable $x^p \in [0,1]$ by using a binary vector of length l the maximal accuracy is $1/2^{l+1}$.

Integer Representations

Instead of using binary strings with cardinality $\chi = 2$, where $\chi \in \{\mathbb{N}^+ \setminus \{0,1\}\}$, higher χ-ary alphabets can also be used for the genotypes. Then, instead of a binary alphabet a χ-ary alphabet is used for the string of length l. Instead of encoding 2^l different individuals with a binary alphabet, we are able to encode χ^l different possibilities. The size of the search space increases from $|\Phi_g| = 2^l$ to $|\Phi_g| = \chi^l$.

For many integer problems, users often prefer to use binary instead of integer representations because schema processing is maximum with binary alphabets when using standard recombination operators (Goldberg 1990b).

Real-valued Representations

When using real-valued representations, the search space Φ_g is defined as $\Phi_g = \mathbb{R}^l$, where l is the length of the real-valued chromosome. When using real-valued representations, researchers often favor mutation-based GEAs like evolution strategies or evolutionary programming. These types of optimization methods mainly use mutation and search through the search space by adding a multivariate zero-mean Gaussian random variable to each variable. In contrast, when using crossover-based GEAs real-valued problems are often represented by using binary representations (see previous paragraph about binary representations).

Real-valued representations can not exclusively be used for encoding real-valued problems, but also for other permutation and combinatorial problems. Trees, schedules, tours, or other combinatorial problems can easily be represented by using real-valued vectors and special genotype-phenotype mappings (see also Sect. 6.4 (LNB encoding) and Sect. 6.5 (NetKeys)).

Messy Representations

In all the previously presented representations, the position of each allele is fixed along the chromosome and only the corresponding value is specified. The first gene-independent representation was proposed by Holland (1975). He proposed the inversion operator which changes the relative order of the alleles in the string. The position of an allele and the corresponding value are coded together as a tuple in a string. This type of representation can be used for binary, integer, and real-valued representations and allows an encoding which is independent of the position of the alleles in the chromosome. Later, Goldberg et al. (1989) used this position-independent representation for the messy genetic algorithm.

Direct Representations

In Sect. 2.1.2, we have seen that a representation is direct if $f_g(x^g) = x^g$. Then, a phenotype is the same as the corresponding genotype and the problem-specific genetic operators are applied directly to the phenotype.

As long as $x^p = x^g$ is either a binary, an integer, or a real-valued string, standard recombination and mutation operators can be used. Then, it is often easy to predict GEA performance by using existing theory. The situation is different if direct representations are used for problems whose phenotypes are not binary, integer, or real-valued. Then, standard recombination and mutation operators can not be used any more. Specialized operators are necessary that allow offspring to inherit important properties from their parents (Radcliffe 1991a; Radcliffe 1991b; Kargupta et al. 1992; Radcliffe 1993a). In general, these operators depend on the specific structure of the phenotypes x^p and must be developed for every optimization problem separately. For more information about direct representations we refer to Chap. 7.

2.2 Genetic and Evolutionary Algorithms

In this section, we introduce genetic and evolutionary algorithms. We illustrate basic principles and outline the basic functionality of GEAs. The schema theorem stated by Holland (1975) explains the performance of selectorecombinative GAs and leads us to the building block hypothesis. The building block hypothesis tells us that short, low-order and highly fit schemata can be recombined to form higher-order schemata and complete strings with high fitness.

2.2.1 Principles

Genetic and evolutionary algorithms were introduced by Holland (1975) and Rechenberg (1973). By imitating basic principles of nature they created optimization algorithms which have successfully been applied to a wide variety of problems. The basic principles of GEAs are derived from the principles of life which were first described by Darwin (1859):

> "Owing to this struggle for life, variations, however slight and from whatever cause proceeding, if they be in any degree profitable to the individuals of a species, in their infinitely complex relations to other organic beings and to their physical conditions of life, will tend to the preservation of such individuals, and will generally be inherited by the offspring. The offspring, also, will thus have a better chance of surviving, for, of the many individuals of any species which are periodically born, but a small number can survive. I have called this principle, by which each slight variation, if useful, is preserved, by the term Natural Selection."

Darwin's ideas about the principles of life can be summarized by the following three basic principles:

- There is a population of individuals with different properties and abilities. An upper limit for the number of individuals in a population exists.
- Nature creates new individuals with similar properties to the existing individuals.
- Promising individuals are selected more often for reproduction by natural selection.

In the following section, we briefly illustrate these three principles. We have seen in Sect. 2.1 that the properties and abilities of an individual which are characterized by its' phenotype are encoded in the genotype. Therefore, based on different genotypes, individuals with different properties exist (Mendel 1866). Because resources are finite, the number of individuals that form a population is limited. If the number of individuals exceeds the existing upper limit, some of the individuals are removed from the population.

The individuals in the population do not remain the same, but change over the generations. New offspring are created which inherit some properties of their parents. These new offspring are not chosen randomly but are somehow similar to their parents. To create the offspring, genetic operators like mutation and recombination are used. Mutation operators change the genotype of an individual slightly, whereas recombination operators combine the genetic information of the parents to create new offspring.

When creating offspring, natural selection more often selects promising individuals for reproduction than low-quality solutions. Highly fit individuals are allowed to create more offspring than inferior individuals. Therefore, inferior individuals are removed from the population after a few generations and have no chance of creating offspring with similar properties. As a result, the average fitness of a population increases over the generations.

In the following paragraphs, we want to describe how the principles of nature were used for the design of genetic and evolutionary algorithms.

2.2.2 Functionality

Genetic and evolutionary algorithms imitate the principles of life outlined in the previous subsection and use it for optimization purposes.

Researchers have proposed many different variants of GEAs in the literature. For illustrating the basic functionality of GEAs we want to use the traditional standard simple genetic algorithm (GA) illustrated by Goldberg (1989c). This type of GEAs uses crossover as the main operator and mutation serves only as background noise. GAs are widely known and well understood. GAs use a constant population of size N, the individuals consist of binary strings with length l, and genetic operators like uniform or n-point crossover are directly applied to the genotypes. The basic functionality of a traditional simple GA is very simple. After randomly creating and evaluating

an initial population, the algorithm iteratively creates new generations. New generations are created by recombining the selected highly fit individuals and applying mutation to the offspring.

- initialize population
  - create initial population
  - evaluate individuals in initial population
- create new populations
  - select fit individuals for reproduction
  - generate offspring with genetic operator crossover
  - mutate offspring
  - evaluate offspring

One specific type of genetic algorithms are selectorecombinative GAs. These types of GAs use only selection and recombination (crossover). No mutation is used. Using selectorecombinative GAs gives us the advantage of being able to investigate the effects of different representations on crossover alone and to eliminate the effects on mutation. This is useful if we use GAs in a way such that they propagate schemata (compare Sect. 2.2.3), and where mutation is only used as additional background noise. When focusing on GEAs where mutation functions as the main search operator, the reader is referred to other work (Rechenberg 1973; Schwefel 1975; Schwefel 1981; Schwefel 1995; Bäck and Schwefel 1995).

In the following paragraphs, we briefly explain the basic elements of a GA. For selecting highly fit individuals for reproduction a large number of different selection schemes have been developed. The most popular are proportionate (Holland 1975) and tournament selection (Goldberg et al. 1989). When using proportionate selection, the expected number of copies an individual has in the next population is proportional to its fitness. The chance of an individual $\boldsymbol{x}_i$ of being selected for recombination is calculated as

$$\frac{f(\boldsymbol{x}_i)}{\sum_{j=1}^{N} f(\boldsymbol{x}_j)},$$

where N denotes the number of individuals in a population. With increasing fitness an individual is chosen more often for reproduction.

When using tournament selection, a tournament between s randomly chosen different individuals is held and the one with the highest fitness is chosen for recombination and added to the mating pool. After N tournaments of size s the mating pool is filled. We have to distinguish between tournament selection with and without replacement. If we perform tournament selection with replacement we choose for every tournament s individuals from all individuals in the population. Then, the mating pool is filled after N tournaments. If we perform a tournament without replacement there are s rounds. In each round we have N/s tournaments and we choose the individuals for a tournament

from those who have not already taken part in a tournament in this round. After all individuals have performed a tournament in one round (after N/s tournaments) the round is over and all individuals are considered again for the next round. Therefore, to completely fill the mating pool s rounds are necessary.

The mating pool consists of all individuals who are chosen for recombination. When using tournament selection, there are no copies of the worst individual, and either an average of s copies (with replacement), or exactly s copies (without replacement) of the best individual in the mating pool. For more information concerning different tournament selection schemes see Bäck et al. (1997, C2) and Sastry and Goldberg (2001).

Crossover operators imitate the principle of sexual reproduction and are applied to the individuals in the mating pool. In many GA implementations, crossover produces two new offspring from two parents by exchanging substrings. The most common crossover operators are one-point (Holland 1975), and uniform crossover (Syswerda 1989). When using one-point crossover, a crossover point $c = \{1, \ldots, l-1\}$ is initially chosen randomly. Two children are then created from the two parents by swapping the substrings. As a result, we get for the parents $\boldsymbol{x}^{p1} = x_1^{p1}, x_2^{p1}, \ldots, x_l^{p1}$ and $\boldsymbol{x}^{p2} = x_1^{p2}, x_2^{p2}, \ldots, x_l^{p2}$ the offspring $\boldsymbol{x}^{o1} = x_1^{p1}, x_2^{p1}, \ldots, x_c^{p1}, x_{c+1}^{p2} \ldots x_l^{p2}$ and $\boldsymbol{x}^{o2} = x_1^{p2}, x_2^{p2}, \ldots, x_c^{p2}, x_{c+1}^{p1} \ldots x_l^{p1}$. When using uniform crossover it is decided independently for every single allele of the offspring from which parent it inherits the value of the allele. In most implementations no parent is preferred and the probability of an offspring to inherit the value of an allele from a specific parent is $p = 1/x$, where x denotes the number of parents that are considered for recombination. For example, when two possible offspring are considered with same probability ($p = 1/2$), we could get as offspring $\boldsymbol{x}^{o1} = x_1^{p1}, x_2^{p1}, x_3^{p2}, \ldots, x_{l-1}^{p1}, x_l^{p2}$ and $\boldsymbol{x}^{o2} = x_1^{p2}, x_2^{p2}, x_3^{p1}, \ldots, x_{l-1}^{p2}, x_l^{p1}$. We see that uniform crossover can also be interpreted as $(l-1)$-point crossover.

Mutation operators should slightly change the genotype of an individual. Mutation operators are important for local search, or if some alleles are lost during a GEA run. By randomly modifying some alleles in the population already lost alleles can be reanimated. The probability of mutation p_m must be selected to be at a low level because otherwise mutation would randomly change too many alleles and the new individual would have nothing in common with its parent. Offspring would be generated almost randomly and genetic search would degrade to random search. In contrast to crossover operators, mutation operators focus more on local search because they can only modify properties of individuals but can not recombine properties from different parents.

2.2.3 Schema Theorem and Building Block Hypothesis

We review explanations for the performance of selectorecombinative genetic algorithms. We start by illustrating the notion of schemata. This is followed

by a brief summary of the schema theorem and a description of the building block hypothesis.

Schemata

Schemata were first proposed by Holland (1975) to model the ability of GEAs to process similarities between bitstrings. A schema $h = (h_1, h_2, \ldots, h_l)$ is defined as a ternary string of length l, where $h_i \in \{0, 1, *\}$. $*$ denotes the "don't care" symbol and tells us that the allele at this position is not fixed. The size or order $o(h)$ of a schema h is defined as the number of fixed positions (0s or 1s) in the string. A position in a schema is fixed if there is either a 0 or a 1 at this position. The defining length $\delta(h)$ of a schema h is defined as the distance between the two outermost fixed bits. The fitness of a schema is defined as the average fitness of all instances of this schema and can be calculated as

$$f(h) = \frac{1}{||h||} \sum_{\boldsymbol{x} \in h} f(\boldsymbol{x}),$$

where $||h||$ is the number of individuals $\boldsymbol{x} \in \Phi_g$ that are an instance of the schema h. The instances of a schema h are all genotypes where $x^g \in h$. For example, $x^g = 01101$ and $x^g = 01100$ are instances of $h = 0 * 1 * *$. The number of individuals that are an instance of a schema h can be calculated as $2^{l-o(h)}$.

For a more detailed explanation of schemata in the context of GEAs the reader is referred to Holland (1975), Goldberg (1989c), or Radcliffe (1997).

Schema Theorem

Based on the notion of schemata, Holland (1975) and De Jong (1975) formulated the schema theorem which describes how the number of instances of a schema h changes over the number of generations t:

$$m(h, t+1) \geq m(h, t) \frac{f(h, t)}{\bar{f}(t)} (1 - p_c \frac{\delta(h)}{l-1} - p_m o(h)),$$

where

- $m(h, t)$ is the number of instances of schema h at generation t,
- $f(h, t)$ is the fitness of the schema h at generation t,
- $\bar{f}(t)$ is the average fitness of the population at generation t,
- $\delta(h)$ is the defining length of schema h,
- p_c is the probability of crossover,
- p_m is the probability of mutation,
- l is the string length,
- $o(h)$ is the order of schema h.

The schema theorem describes how the number of copies that are given to a schema h depends on selection, crossover and mutation, when using a standard GA with proportionate selection, one-point crossover, and bit-flipping mutation. Selection favors a schema if the fitness of the schema is above the average fitness of the population ($f(h,t) > \bar{f}(t)$). When using crossover the defining length $\delta(h)$ of a schema must be small because otherwise one-point crossover frequently disrupts long schemata. The bit-flipping mutation operator favors low order schemata because with increasing $o(h)$ the number of schemata which are destroyed increases.

The main contribution of the schema theorem is that schemata, which fitness is above average ($f(h) > \bar{f}$), which have a short defining length $\delta(h)$, and which are of low order $o(h)$, receive exponentially increasing trials in subsequent generations. The theorem describes the hurdle between selection, which preserves highly fit schemata, and crossover and mutation which both destroy schemata of large order or defining-length.

This observation brings us to the concept of *building blocks* (BBs). Goldberg (1989c, p. 20 and p. 41) defined buildings blocks as "highly fit, short-defining-length schemata" that "are propagated generation to generation by giving exponentially increasing samples to the observed best". The notion of building blocks is frequently used in the literature but rarely defined. In general, a building block can be described as a solution to a sub-problem that can be expressed as a schema. A thus-like schema has high fitness and its size is smaller than the length of the string. By combining BBs of lower order, a GA can form high-quality over-all solutions. Using the notion of genes we can interpret BBs as genes. A gene consists of one or more alleles and can be described as a schema with high fitness. The alleles in a chromosome can be separated (decomposed) into genes which do not interact with each other and which determine one specific phenotypic property of an individual like hair or eye color. We see that by using building blocks we can describe – with the help of the schema theorem – how GAs can solve an optimization problem. If the sub-solutions to a problem (the BBs) are short (low $\delta(h)$) and of low order (low $o(h)$), then the number of correct sub-solutions increases over the generations and the problem can easily be solved by a GA.

The schema theorem and the concept of building blocks have attracted a lot of critical comments from various researchers (Radcliffe 1991b; Vose 1991; Vose 1999). The comments are mainly based on the observation that the schema theorem does not always explain the observed behavior of GEAs as it neglects the stochastic and dynamic nature of the genetic search. Different approaches have been presented to develop and extend the schema theorem. Altenberg (1994) related the schema theorem to Price's theorem (Price 1970), which can be viewed as a more general formulation of the schema theorem. Radcliffe introduced the concept of forma (compare Sect. 2.4.2) which allows us to introduce schemata not only for binary strings, but also for general genotypes. Poli et al. developed exact schema theorems in the context of genetic programming which could also be applied to general GEAs (Stephens

and Waelbroeck 1999; Poli 2001a; Poli 2001b). An in depth survey of the critical comments on the schema theorem including appropriate extensions and later developments can be found in Reeves and Rowe (2003, Sect. 3).

Building Block Hypothesis

Using the definition of building blocks as being highly fit solutions to sub-problems, the *building block hypothesis* can be formulated. It describes the processing of building blocks and is based on the quasi-decomposability of a problem (Goldberg 1989c, page 41):

> "Short, low order, and highly fit schemata are sampled, recombined, and resampled to form strings of potentially higher fitness."

The building block hypothesis basically states that GEAs mainly work due to their ability to propagate building blocks. By combining schemata of lower order which are highly fit, a GEA can construct overall good solutions.

The building block hypothesis can be used for explaining the high performance of GEAs in many real-world applications. It basically says that a schemata processing GEA performs well, if the problem it is applied to is quasi-decomposable, that means the overall problem can be separated into smaller sub-problems. If the juxtaposition of smaller, highly fit, partial solutions (building blocks) does not result in good solutions, GEAs would fail in many real-world problems. Only by decomposing the overall problem into many smaller sub-problems, solving these sub-problems separately, and combining the good solutions, can a GEA find good solutions to the overall optimization problem (Goldberg 2002).

This observation raises the question of why the approach of separating complex problems into smaller ones and solving the smaller problems to optimality is so successful. The answer can be found in the structure of the problems themselves. Many of the problems in the real world are somehow decomposable, because otherwise all our design and optimization methods which try to decompose complex problems could not work properly. A look in the past reveals that approaching real-world problems in the outlined way has resulted in quite interesting results. Not only do human designers or engineers use the property of many complex real-world problems to be decomposable, but nature itself. Most living organisms are not just one complex system where each part interacts with all others, but they consist of various separable subsystems for sensing, movement, reproduction, or communication. By optimizing the subsystems separately, and combining efficient subsystems, nature is able to create complex organisms with surprising abilities.

Therefore, if we assume that many of the problems in the real world can be solved by decomposing them into smaller sub-problems, we can explain the good results obtained by using GEAs for real-world problems. GEAs perform well because they try, in analogy to human intuition, to decompose the overall problem into smaller parts (the building blocks), solve the smaller

sub-problems, and combine the good solutions. A problem can be properly decomposed by identifying the interdependencies between the different alleles. The purpose of the genetic operators is to decompose the problem by detecting which alleles in the chromosome influence each other, to solve the smaller problems efficiently, and to combine the sub-solutions (Harik and Goldberg 1996; Harik and Goldberg 1996).

2.3 Problem Difficulty

Previous work has shown that representations influence the behavior and performance of GEAs (Goldberg 1989c; Liepins and Vose 1990). The results revealed that when using specific representations some problems become easier, whereas other problems become more difficult to solve for GEAs. To be able to systematically investigate how representations influence GEA performance, a measurement of problem difficulty is necessary. With the help of a difficulty measurement, it can be determined how representations change the complexity and difficulty of a problem. However, a problem does not have the same difficulty for all types of optimization algorithms, but difficulty always depends on the optimization method used. Therefore, focusing on selectorecombinative GEAs also determines the reasons of problem difficulty: building blocks.

Consequently, in Sect. 2.3.1 we discuss reasons of problem difficulty and illustrate for different types of optimization methods that different reasons for problem difficulty exist. As we focus on selectorecombinative GEAs and assume that these types of GEAs process building blocks, we decompose problem difficulty with respect to BBs. This is followed in Sect. 2.3.2 by an illustration of different measurements of problem difficulty. The measurements of problem difficulty are based on the used optimization method. Because we focus in this work on schemata and BBs, we later use the schemata analysis as a measurement of problem difficulty.

2.3.1 Reasons for Problem Difficulty

One of the first approaches to the question of what makes problems difficult for GEAs, was the study of deceptive problems by Goldberg (1987). His studies were mainly based on the work of Bethke (1981). These early statements about deceptive problems were the origin of a discussion about the reasons of problem difficulty in the context of genetic and evolutionary algorithms. Searching for reasons of problem difficulty means investigating what makes problems difficult for GEAs.

Researchers recognized that there are other possible reasons of problem difficulty besides deception. Based on the structure of the fitness landscape (Weinberger 1990; Manderick et al. 1991), the correlation between the fitness of individuals describes how difficult a specific problem is to solve for GEAs. By the modality of a problem, which is more popular for mutation-based

search methods, problems can be classified into easy unimodal problems (there is only one local optimum), and difficult multi-modal problems (there are many local optima). Another reason for difficulty is found to be epistasis, which is also known as the linear separability of a problem. This describes the interference between the alleles in a string and measures how well a problem can be decomposed into smaller sub-problems (Holland 1975; Davidor 1989; Davidor 1991; Naudts et al. 1997). A final reason for problem difficulty is additional noise which makes most problems more difficult to solve for GEAs.

Many of these approaches are not focused on schemata-processing selectorecombinative GEAs. Goldberg (2002) presented an approach of understanding problem difficulty based on the schema theorem and the building block hypothesis. He viewed problem difficulty for selectorecombinative GEAs as a matter of building blocks and decomposed it into

- difficulty within a building block (intra-BB difficulty),
- difficulty between building blocks (inter-BB difficulty), and
- difficulty outside of building blocks (extra-BB difficulty).

This decomposition of problem difficulty assumes that difficult problems are building block challenging. In the following paragraphs, we briefly discuss these three aspects of BB-complexity.

If we count the number of schemata of order $o(h) = k$ that have the same fixed positions, there are 2^k different competing schemata. Based on their fitness, the different schemata compete against each other and GEAs should increase the number of the high-quality schemata. Identifying the high quality schemata and propagating them properly is the main difficulty of intra-BB difficulty. Goldberg measures intra-BB difficulty with the deceptiveness of a problem. Deceptive problems (Goldberg 1987) are most difficult to solve for GEAs because GEAs are led by the fitness landscape to a deceptive attractor which has maximum distance to the optimum. To reliably solve difficult, for example deceptive problems, GEAs must increase the number of copies of the best BB by giving enough copies to them.

One basic assumption of the schema theorem is that a problem can be decomposed into smaller sub-problems. GEAs solve these smaller sub-problems in parallel and try to identify the correct BBs. In general, the contributions of different BBs to the fitness function are not uniform and there can be interdependencies between the different BBs. Because different BBs can have different contributions to the fitness of an individual, the loss of low salient BBs during a GEA run is one of the major problems of inter-BB difficulty (compare Sect. 3.2.3). Furthermore, a problem can often not be decomposed into completely separate and independent sub-problems, but there are still interdependencies between the different BBs which are an additional source of inter-BB difficulty.

Even for selectorecombinative GEAs there is a world outside the world of schemata and BBs. Sources of extra-BB difficulty like noise have an additional influence on the performance of GEAs because selection is based on the fitness

of the individuals. Additional, non-deterministic noise randomly modifies the fitness values of the individuals. Therefore, selection decisions are no longer based on the quality of the solutions (and of course the BBs) but on stochastic variance. A similar problem occurs if the evaluation of the individuals is non-stationary. Evaluating fitness in a non-stationary way means that individuals have a different fitness at different moments in time.

In the remainder of the subsection, we discuss that reasons of problem difficulty must be seen in the context of a specific optimization method. If different optimization methods are used for the same problem then there are different reasons of problem difficulty. As a result, there is no general problem difficulty for all types of optimization methods but we must independently identify for each optimization method the reasons of problem difficulty. We illustrate how problem difficulty depends on the used optimization method with two small examples.

When using random search, the discussion of problem complexity is obsolete. During random search new individuals are chosen randomly and no prior information about the structure of the problem or previous search steps is used. As a result, all possible types of problems have the same difficulty. Although measurements of problem complexity, like correlation analysis or the analysis of intra-, inter-, or extra-BB difficulty, lead us to believe that some problems are easier to solve than others, there are no easy or difficult problems. Independently of the complexity of a problem, random search always needs on average the same number x of fitness evaluations for finding the optimal solution. All problems have the same difficulty regarding random search, and to search for reasons of problem difficulty makes no sense.

When comparing crossover- and mutation-based evolutionary search methods, different reasons of problem complexity exist. From the schema theorem we know that selectorecombinative GEAs propagate schemata and BBs. Therefore, BBs are the main source of complexity for these types of GEAs. Problems are easy for selectorecombinative GEAs if the problem can be properly decomposed into smaller sub-solutions (the building blocks) and the intra-, and inter-BB difficulty is low. However, when using mutation-based approaches like evolution strategies (Rechenberg 1973; Schwefel 1975), these reasons for problem difficulty are not relevant any more. Evolution strategies perform well if the structure of the solution space guides the population to the optimum (compare the good performance of evolution strategies on unimodal optimization problems). Problem complexity is not based on BBs, but more on the structure of the fitness landscape. To use the notion of BBs for mutation-based optimization methods makes no sense because they propagate no schemata.

We have illustrated how the difficulty of a problem for selectorecombinative GEAs can be decomposed into intra-BB, inter-BB, and extra BB-difficulty. The decomposition is based on the assumption that GEAs decompose problems and work with schemata and BBs. The proposed BB-based reasons of problem difficulty can be used for selectorecombinative GEAs but can not

be applied to other optimization methods like evolution strategies or random search.

2.3.2 Measurements of Problem Difficulty

In the previous section, we discussed the reasons for problem difficulty. In the following paragraphs, we describe some measurements of problem difficulty.

To investigate how different representations influence the performance of GEAs, a measurement of problem difficulty is necessary. Based on the different reasons for problem difficulty which exist for different types of optimization methods, we discuss some common measurements of problem difficulty:

- Correlation analysis,
- polynomial decomposition,
- Walsh coefficients, and
- schemata analysis.

These four measurements of problem difficulty are widely used in the GEA literature for measuring different types of problem difficulty (Goldberg 1989b; Goldberg 1992; Radcliffe 1993a; Horn 1995; Jones and Forrest 1995). For an overview see Bäck et al. (1997, Chap. B2.7) or Reeves and Rowe (2003). The specific properties of the four measurements are briefly discussed in the following paragraphs.

Correlation analysis is based on the assumption that the high and low quality solutions are grouped together and that GEAs can use information about individuals whose genotypes are very similar for generating new offspring. Therefore, problems are easy if the structure of the search space guides the search to the high quality solutions. Consequently, correlation analysis is a proper measurement for the difficulty of a problem when using mutation-based search approaches. Correlation analysis exploits the fitness between neighboring individuals of the search space as well as the correlation of the fitness between parents and their offspring (For a summary see Deb et al. (1997)). The most common measurements for distance correlation are the autocorrelation function of the fitness landscape (Weinberger 1990), the fitness correlation coefficients of genetic operators (Manderick et al. 1991), and the fitness-distance correlation (Jones 1995; Jones and Forrest 1995; Altenberg 1997).

The linearity of an optimization problem can be measured by the polynomial decomposition of the problem. Each function f defined on $\Phi_g = \{0,1\}^l$ can be decomposed in the form

$$f(\boldsymbol{x}) = \sum_{N \subset \{1,\ldots,l\}} \alpha_N \prod_{n \in N} \boldsymbol{e}_n^T \boldsymbol{x},$$

where the vector $\boldsymbol{e}_n$ contains 1 in the nth column and 0 elsewhere, T denotes transpose, and the α_N are the coefficients (Liepins and Vose 1991). Regarding

the vector $\boldsymbol{x}$ having components $x_1, \ldots, x_l$, we may view f as a polynomial in the variables $x_1, \ldots, x_l$. The coefficients α_i describe the non-linearity of the problem. If there are high order α_N in the decomposition of the problem, the function is highly nonlinear. If the decomposition of a problem only has order 1 coefficients, then the problem is linear and easy for GEAs. It is possible to determine the maximum non-linearity of $f(\boldsymbol{x})$ by its highest polynomial coefficients. The higher the order of the α_i, the more nonlinear the problem is.

There is some correlation between the non-linearity of a problem and the difficulty of a problem for selectorecombinative GEAs (Mason 1995), but the order of non-linearity can only give an upper limit of the problem difficulty. As illustrated in the following example there could be high order α_i although the problem still remains easy for a GA. The function

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} = 00, \\ 2 & \text{if } \boldsymbol{x} = 01, \\ 4 & \text{if } \boldsymbol{x} = 10, \\ 10 & \text{if } \boldsymbol{x} = 11, \end{cases} \tag{2.1}$$

could be decomposed in $f(\boldsymbol{x}) = \alpha_0 + \alpha_1 x_0 + \alpha_2 x_1 + \alpha_3 x_0 x_1 = 1 + x_0 + 3x_1 + 5x_0 x_1$. The problem is easy for selectorecombinative GEAs (all BBs are of order $k = 1$), but nonlinear.

Instead of decomposing a problem into its polynomial coefficients, it can also be decomposed into the corresponding Walsh coefficients. The Walsh transformation is analogous to the discrete Fourier transformation but for functions whose domain is a bitstring. Every real-valued function $f : \Phi_g \rightarrow \mathbb{R}$ over an bitstring of length l, can be expressed as a weighted sum of a set of 2^l orthogonal functions called Walsh functions:

$$f(\boldsymbol{x}) = \sum_{j=0}^{2^l-1} w_j \ \psi_j(\boldsymbol{x}),$$

where the Walsh functions are denoted $\psi_j : \Phi_g \rightarrow \{-1, 1\}$. The weights $w_j \in \mathbb{R}$ are called Walsh coefficients. The indices of both Walsh functions and coefficients are the numerical equivalent of the binary string $\boldsymbol{j}$. The jth Walsh function is defined as:

$$\psi_j(\boldsymbol{x}) = (-1)^{bc(\boldsymbol{j} \wedge \boldsymbol{x})},$$

with $\boldsymbol{x}, \boldsymbol{j}$ are binary strings and elements of Φ_g, $\wedge$ denotes the bitwise logical and, and $bc(\boldsymbol{x})$ is the number of one bits in $\boldsymbol{x}$. For a more detailed explanation the reader is referred elsewhere (Goldberg 1989a; Goldberg 1989b; Vose and Wright 1998a; Vose and Wright 1998b). The Walsh coefficients can be computed by the Walsh transformation:

$$w_j = \frac{1}{2^l} \sum_{i=0}^{2^l-1} f(i)\psi_j(i).$$

The coefficients w_j measure the contribution to the fitness function by the interaction of the bits indicated by the positions of the 1's in j. The larger the value of j, the higher the order of the interactions between the bits in j is. For example, w_{001} measures the linear contribution to f associated with bit position 2. w_{111} measures the nonlinear interaction between all three bits. Any function f over an l bit space can be represented as a weighted sum of all possible 2^l bit interaction functions ψ_j.

Walsh coefficients are an important feature in measuring the problem difficulty for GEAs (Goldberg 1989a; Oei 1992; Goldberg 1992; Reeves and Wright 1994). It was shown that problems are easy for GEAs if the Walsh coefficients are of order 1. Furthermore, difficult problems tend to have high order Walsh coefficients, but nevertheless the Walsh coefficients do not give us an exact measurement of problem complexity. The highest order of the coefficient w_i can only give an upper limit of the problem complexity. Therefore, Walsh coefficients show the same behavior as polynomials. This behavior is expected as it has already been shown that Walsh functions are polynomials (Goldberg 1989a; Goldberg 1989b; Liepins and Vose 1991). The insufficient measurement of problem difficulty for selectorecombinative GEAs can be illustrated with the earlier example (2.1). The Walsh coefficients for the former example are $w = \{4.25, -1.75, -2.75, 1.25\}$. Although the problem is quite simple, there are high order Walsh terms.

If we assume that selectorecombinative GEAs process schemata and BBs, then the most natural and direct way to measure problem complexity is to analyze the size and length of the building blocks in the problem. If we assume that GEAs process building blocks, the intra-BB difficulty of a problem can be measured by the maximum length $\delta(h)$ and size $k = o(h)$ of the BBs h (Goldberg 1989c).

A problem is denoted to be deceptive of order k_{max} if for $k < k_{max}$ all schemata that contain parts of the best solution have lower fitness than their competitors (Deb and Goldberg 1994). Schemata are competitors if they have the same fixed positions. An example for competing schemata of size $k = 2$ for a bitstring of length $l = 4$ are $h = 0 * 0*$, $h = 0 * 1*$, $h = 1 * 0*$, and $h = 1 * 1*$. Therefore, the highest order k_{max} of the schemata that are not misleading determines the complexity of a problem for selectorecombinative GEAs. The higher the maximum order k_{max} of the schemata, the more difficult the problem is to solve for GEAs.

The average fitness of the schemata for the brief example illustrated in (2.1) is shown in Table 2.1. All schemata that contain a part of the optimal

order	2			1		0
schema	11			1*	*1	**
fitness	**10**			**7**	**6**	**4.25**
schema	01	10	00	0*	*0	
fitness	2	4	1	1.5	2.5	

Table 2.1. Average schema fitness for example described by (2.1)

solution are above average and better than their competitors. Calculating the deceptiveness of the problem based on the fitness of the schemata correctly classifies this problem to be very easy. Since we analyze the influence of representations on selectorecombinative GEAs, and we assume that these types of GEAs process schemata, schema fitness averages are used to measure problem difficulty in the remainder of this work. Problems of length l are defined to be *fully easy* if $k_{max} = 1$, and to be *fully difficult* (compare Goldberg (1992)) if $k_{max} = l$. Therefore, when using selectorecombinative GEAs fully easy problems are the most easy problems, whereas fully difficult problems are the most difficult problems.

2.4 Existing Recommendations for the Design of Efficient Representations

Although the application of GEAs to optimization problems is not possible without using representations, mainly intuitive knowledge exists about how to choose proper representations. Up till now, there is no proven theory regarding the influence of representations on the performance of GEAs. To help users with the difficult task of finding good representations, some researchers have made recommendations for the design of efficient representations over the last few years. In this section, we review some recommendations which are important from the authors point of view. For a more detailed overview, the reader is referred to Ronald (1997) and Radcliffe (1991b).

We start in Sect. 2.4.1 with the principle of meaningful building blocks and minimal alphabets which were proposed by Goldberg (1989c). About the same time, Radcliffe developed the concept of forma which is presented in Sect. 2.4.2. Some years later, Palmer (1994) presented more specific guidelines about proper design of representations (compare Sect. 2.4.3). Finally, we illustrate in Sect. 2.4.4 the recommendations made by Ronald (1997).

2.4.1 Goldberg's Meaningful Building Blocks and Minimal Alphabets

Some of the first recommendations for the construction of representations were made by Goldberg (1989c). He proposed the principle of minimal alphabets and of meaningful building blocks.

It is known that the design of an encoding has a strong impact on the performance of a genetic algorithm and should be chosen carefully (compare Coli and Palazzari (1995a), Ronald (1997), and Albuquerque et al. (2000)). Goldberg (1989c, p. 80) proposed two basic design principles for encodings:

- Principle of meaningful building blocks: The schemata should be short, of low order, and relatively unrelated to schemata over other fixed positions.

- Principle of minimal alphabets: The alphabet of the encoding should be as small as possible while still allowing a natural representation of solutions.

The principle of meaningful building blocks is directly motivated by the schema theorem (see Sect. 2.2.3). If schemata are highly fit, short, and of low order, then their numbers exponentially increase over the generations. If the high-quality schemata are long or of high order, they are disrupted by crossover and mutation and they can not be propagated properly by GEAs. Consequently, representations should modify the complexity of a problem in such a way that phenotypically long or high order BBs become genotypically short, and of low order. Then, the problem becomes easier for selectorecombinative GEAs.

The principle of minimal alphabets tells us to increase the potential number of schemata by reducing the cardinality of the alphabet. When using minimal alphabets the number of possible schemata is maximal. This is the reason why Goldberg advises us to use bitstring representations, because high quality schemata are more difficult to find when using alphabets of higher cardinality (Goldberg 1989c, pp. 80ff). But of course we have a trade-off between the low cardinality of an alphabet and the natural expression of the problem. Therefore, sometimes a higher cardinality of the alphabet could be helpful for GEAs (Goldberg 1991b).

Goldberg's two design principles of representations are based on the assumption that GEAs process schemata and BBs, but both principles are very abstract and general, and do not provide the user with exact and applicable guidelines.

2.4.2 Radcliffe's Formae and Equivalence Classes

The principles from Goldberg, which are based on schemata, are made for binary representations and selectorecombinative GAs. Therefore, they are not useful in the design of proper representations when using non-binary genotypes, or other types of search paradigms like evolution strategies or evolutionary programming as these search methods do not process schemata. Consequently, Radcliffe extended the notion of schemata and introduced the more general principle of *forma*, which describes general subsets of the search space (Radcliffe 1991b; Radcliffe 1991a; Radcliffe 1992; Radcliffe 1993b; Radcliffe and Surry 1994; Radcliffe 1994). Formae are defined as equivalence classes that are induced by a set of equivalence relations. Any possible solution of an optimization problem can be identified by specifying the equivalence class to which it belongs for each of the equivalence relations. For example, if we have a search space of faces (Surry and Radcliffe 1996), basic equivalence relations might be "same hair color" or "same eye color", which would induce the formae "red hair", "dark hair", "green eyes", etc. Formae of higher order like "red hair and green eyes" are then constructed by composing simple formae. The search space, which includes all possible faces, can be constructed

with strings of alleles that represent the different formae. For the definition of formae the structure of the phenotypes is important. For example, for binary problems possible formae would be "bit i is equal to one/zero". When encoding tree structures, possible basic formae would be "contains link from node i to node j". Based on these basic formae, we develop in Sect. 7.1 a direct representation for trees with appropriate search operators.

An unsolved problem is to find appropriate equivalences for the different instances of a given problem. From the equivalences the genotypic search space Φ_g and the genotype-phenotype mapping f_g can be constructed. Usually, a solution is encoded as a string of alleles, each of which indicates that the solution satisfies a particular equivalence. Radcliffe (1991a) proposed several design principles for creating appropriate equivalences for a given problem. The most important design principle is that the generated formae should group together solutions of related fitness (Radcliffe and Surry 1994), in order to create a fitness landscape or structure of the search space that can be exploited by search operators. Search operators are constructed based on the defined formae.

Radcliffe recognized that the genotypic search space, the genotype-phenotype mapping, and the search operators belong together and their design can not be separated from each other (Radcliffe 1992). For the development of appropriate search operators that are based on pre-defined formae he formulated the following four design principles (Radcliffe 1991a; Radcliffe 1994):

- **Respect**: Offspring produced by recombination should be members of all formae to which both their parents belong. This means for the "face example" that offspring should have red hair and green eyes if both parents have red hair and green eyes.
- **Transmission**: An offspring should be equivalent to one of its parents under each of the basic equivalence relations. This means that every gene should be set to an allele which is taken from one of the parents. If one parent has dark hair and the other red hair, then the offspring has either dark or red hair.
- **Assortment**: An offspring can be formed with any compatible characteristics taken from the parents. Assortment is necessary as some combinations of equivalence relations may be infeasible. This means for example, that the offspring inherits the dark hair from the first parent and the blue eyes from the second parent only if dark hair and blue eyes are compatible. Otherwise, one of the alleles is set randomly to a feasible value.
- **Ergodicity**: The iterative use of search operators allows us to reach any point in the search space from all possible starting solutions.

The recommendations from Radcliffe illustrate nicely that representations and search operators depend on each other and cannot be designed independently. He developed a consistent concept on how to design efficient GEAs once appropriate equivalence classes (formae) are defined. However, the finding of appropriate equivalence classes, which is equivalent to defining the genotypic

search space and the genotype-phenotype mapping, is often difficult and remains an unsolved problem.

2.4.3 Palmer's Tree Encoding Issues

Palmer analyzed properties of tree representations (Palmer 1994). The recommendations he gave for the design of tree representations can also be applied to other types of representations. Palmer proposed the following representation issues:

- An encoding should be able to represent all possible phenotypes.
- An encoding should be unbiased in the sense that all possible individuals are equally represented in the set of all possible genotypic individuals.
- An encoding should encode no infeasible solutions.
- The decoding of the phenotype from the genotype should be easy.
- An encoding should possess locality. Small changes in the genotype should result in small changes in the phenotype.

Although Palmer formulated the design issues based mainly on intuition rather than on theoretical investigation, the guidelines can advantageously be used for the design of proper representations. For a more detailed description and discussion of these representation issues in the context of tree network representations, the reader is referred to Sect. 6.1.6. We will also see later in this work that the encoding of infeasible solutions (Sect. 6.3.2), a bias of the individuals (Sect. 6.4.2), or low locality of an encoding (Sect. 8.1.3) is not always necessarily disadvantageous for the performance of GEAs.

2.4.4 Ronald's Representational Redundancy

A few years ago, Ronald presented a survey of encoding issues (Ronald 1997; Ronald 1995). Representations should be chosen according to the following guidelines:

- Encodings should be adjusted to a set of genetic operators in a way that the building blocks are preserved from the parents to the offspring (Fox and McMahon 1991).
- Encodings should minimize epistasis (Beasley et al. 1993).
- Feasible solutions should be preferred.
- The problem should be represented at the correct level of abstraction.
- Encodings should exploit an appropriate genotype-phenotype mapping process if a simple mapping to the phenotype is not possible.
- Isomorphic forms, where the phenotype of an individual is encoded with more than one genotype, should not be used.

Many of the representation issues can be put down to the principles of representations illustrated in Sect. 2.4.1. The design issue concerning isomorphic forms will be discussed in Sect. 3.1. The results will show that by using isomorphic or redundant representations the performance of GEAs can easily be increased.

3

Three Elements of a Theory of Representations

In this chapter, we study an often ignored aspect of heuristic optimization, namely the theory of representations for genetic and evolutionary algorithms. Although the importance of choosing proper representations for the performance of genetic and evolutionary algorithms is already recognized (Caruana and Schaffer 1988; Goldberg 1989c; Liepins and Vose 1990; Ronald et al. 1995; Radcliffe 1991a; Coli and Palazzari 1995b; Ronald 1997; Albuquerque et al. 2000; Kargupta 2000a; Schnier and Yao 2000; Hinterding 2000), we are still far from a complete theory of representations.

Due to the fact that developing a general theory of representations is a formidable challenge, we decompose this task into smaller parts. We start by presenting three elements of representation theory which are the basis of the time-quality framework of representations which we present in Chap. 4. Namely, we focus on redundancy, scaling of alleles, and locality which describes the modification of distances between corresponding genotypes and phenotypes. We present theoretical models for these three aspects of representation theory and show how these properties of representations affect the performance of GEAs.

The following paragraphs discuss these three aspects of representation theory. A representation is denoted to be redundant if the number of genotypes is higher than the number of phenotypes. Therefore, for a redundant representation, a phenotype is represented on average by more than one genotype. Investigating redundancy more closely, we have to distinguish between synonymously and non-synonymously redundant representations. Non-synonymously redundant representations do not allow genetic operators to work properly and therefore reduce the efficiency of evolutionary search. When using synonymously redundant representations, GEA performance mainly depends on the change of the initial supply. Based on this observation models can be developed that allow us to determine the necessary population size and the number of generations for solving a problem. Redundant representations are uniformly redundant if each phenotype is on average represented by the same number of genotypes. Theoretical and empirical results show that represen-

tations that are synonymously and uniformly redundant do not change the behavior of GEAs. Furthermore, the results show that representations that are synonymously and non-uniformly redundant can only be used advantageously if there exists some a-priori information about the optimal solution.

When assigning phenotypes to genotypes, a representation f_g can change the importance of the alleles. For example, if a phenotype is a list of integers, all alleles (integers) are equally relevant for calculating the fitness of a phenotype. However, when encoding the phenotypic integers using binary strings, the contributions of the genotypic bits to the construction of the phenotypes and to the calculation of the corresponding fitness values are no longer equal as some genotypic bits are more relevant than others. By substituting alleles with building blocks (BBs), the order of scaling of a representation describes how different the contributions of the genotypic BBs are to the construction of the phenotypes. It is well known that if the BBs are uniformly scaled, GEAs solve all BBs implicitly in parallel. In contrast, for non-uniformly scaled BBs, domino convergence occurs and the BBs are solved sequentially starting with the most salient BB (Thierens 1995). As a result, the convergence time increases and the search is affected more strongly by genetic drift. Lower salient alleles are not opposed to selection pressure unless they are affected by the solving process. Therefore, some of the lower salient alleles could loose their diversity and are randomly fixed. To model more exactly the effects of non-uniformly scaled representations on GEA performance, we extend the work of Thierens and Goldberg (1993) and present a more general theory of non-uniformly scaled encodings. It allows us to more accurately predict the performance of GEAs using non-uniformly scaled representations under the influence of genetic drift.

In general, the used representation f_g should have no influence on the ability of GEAs to solve easy problems. However, previous work (Liepins and Vose 1990) has shown that representations can easily change the difficulty of an optimization problem. The difficulty of an optimization problem is determined by the mapping f_p that assigns a fitness value to each phenotype. Therefore, by the use of representations, fully easy (compare Sect. 2.3) problems can become fully difficult, and vice versa. Section 3.3 reveals that the locality of a representation determines whether the difficulty of a problem is changed by the representation. The locality of a representation describes how well the distances between individuals are preserved when mapping the genotypes on the phenotypes by the representation f_g. The genotypic distances depend on the type of the used search operator and the phenotypic distances are determined by the character of the optimization problem. We illustrate that high-locality representations, where neighboring phenotypes correspond to neighboring genotypes, preserve problem difficulty and allow GEAs to solve easy problems more reliably. Furthermore, we are able to show both theoretically and empirically, that by using representations where the locality is not preserved, fully easy problems become more difficult, whereas fully difficult problems become easier. Therefore, if our aim is to reliably solve problems

of bounded difficulty, we demand high-quality representations to have low locality as only this guarantees that the difficulty of the problem f_p remains unchanged. Finally, we discuss why, in general, it is not possible to create representations that both, preserve complexity for easy problems, and reduce complexity for difficult problems.

Section 3.1 shows how the usage of redundant encodings affects genetic search. Based on the Gambler's ruin model (Harik et al. 1997), we develop a quantitative model of redundancy and verify it empirically for the trivial voting mapping, which is a synonymously redundant encoding. In Sect. 3.2, we show how the behavior of GEAs changes for exponentially scaled encodings by using the existing models of genetic drift and population sizing. We use two different drift models and develop models for the necessary population size and the convergence time that allows us to predict the solution quality more accurately than the previous models. To verify the theoretical models, we present an empirical investigation into the performance of GEAs using exponentially scaled representations. Section 3.3 shows that only representations with perfect locality guarantee that phenotypically easy problems remain genotypically easy and can still be solved using GEAs. If the locality of a representation is not perfect, the size and length of the BBs can be different for the genotypes and phenotypes and the complexity of the problem is changed. Fully easy problems can only become more difficult, and fully difficult problems can only become easier to solve for GEAs. The chapter ends with concluding remarks.

3.1 Redundancy

This section provides the first of three elements of a theory of representations. It identifies redundancy to be important for the design of representations, distinguishes between synonymously and non-synonymously representations, and uses existing complexity models to characterize the effect of redundancy on encodings. Furthermore, it presents theoretical models on how the population size, run duration and overall problem difficulty is influenced by synonymously redundant encodings. The model is used for the analysis of the trivial voting mapping, which is a synonymously redundant encoding.

3.1.1 Redundant Representations and Neutral Networks

Information theory provides us with a measurement of information. The information content[1] (measured in Bits) of a sequence is defined as the number of bits required to represent a given number of s possibilities using an optimal encoding (Shannon 1948; Shannon and Weaver 1949). It is calculated as $\log_2(s)$. Redundant encodings are less efficient codings that require more bits

[1] other notations are information, self-information, entropy, or Shannon entropy.

to represent the information but do not increase the amount of information represented.

For encoding one Bit of information content (for example the two possibilities 0 and 1) a binary string of at least length one is necessary (one bit). However, it is also possible to encode one Bit of information content using a bitstring of length $l > 1$. Then, more than one bit of the bitstring encodes one Bit of information, and the representation becomes redundant. We want to emphasize that it is important to distinguish between the amount of information (**B**it) that should be represented and the number of **b**its in a string that are used to represent the information. Redundant representations are encodings where the amount of encoded information (in Bit) is lower than the used number of bits. This means, that such encodings use a higher number of alleles for encoding phenotypic information in the genotype than is necessary for constructing the phenotype. Although the practice of redundancy has steadily increased over the last few years, there is little theory regarding the influence of redundant representations on the performance of GEAs.

Natural Selection and Neutral Theory: Different Concepts for Explaining Evolution

Examining the use of redundant representations reveals that redundant representations are not solely an invention of evolutionary computation researchers, but are commonly used in nature for the encoding of genetic information. Currently, in biology different opinions exist regarding the basic concepts that underly the process of evolution in nature and the role of representations therein. *Darwinism* goes back to Darwin (1859) and assumes that *natural selection* is the driving force of evolution (compare Mayr (1991)) and that random genetic drift is unimportant. Randomly advantageous mutations are fixed due to natural selection and can then be propagated from generation to generation. Genetic changes are a result of selection combined with variation operators such as crossover and random mutations. During the process of evolution the variation operators sometimes result in fitter individuals, which gradually replace less-fit individuals in the population.

The theory of natural selection has been extended and modified by the *neutral theory* which was proposed by Kimura (1983). It assumes that the driving force of molecular evolution is the random fixation of neutral mutations rather than the fixation of advantageous mutations by natural selection. Kimura observed that in nature the number of different genotypes which store the genetic material of an individual greatly exceeds the number of different phenotypes which determine the outward appearance. Therefore, the representation which describes how the genotypes are assigned to the phenotypes must be redundant, and neutral mutations become possible. A mutation is neutral if its application to a genotype does not result in a change of the corresponding phenotype. Because large parts of the genotype have no actual effect on the phenotype, evolution can use them as a store for genetic

information that was necessary for survival in the past, and as a playground for developing new properties of the individual that could be advantageous in the future. Neutral mutations are the tool for designing these new properties without interfering with the current phenotype. Although most of the mutations are neutral, some sometimes have an effect on the phenotype and bring new genetic material which was developed by neutral mutations into life. The neutral theory was highly disputed shortly after its formulation and the relative importance of neutral mutation and selection is still unclear. However, the importance of random genetic changes has generally been accepted in population genetics during the last few years.

Redundant Representations in Evolutionary Computation Research

Following the work of Kimura, some biological studies (Huynen et al. 1996; Huynen 1996; Schuster 1997; Reidys and Stadler 1998) focused on the neutral theory. These studies showed that the connectivity between fitness landscapes can be increased by the introduction of redundant representations and neutral mutations. Different genotypes which are assigned to the same phenotype (neutral sets) allow a population to move through the search space more easily and to find new advantageous areas of the search space that would not have been accessible without neutral mutations. Surprisingly, the neutral theory became even more popular in the field of genetic and evolutionary computation (Banzhaf 1994; Dasgupta 1995). There is great interest in how redundant representations and neutral search spaces influence the behavior, and especially the evolvability of GEAs (Barnett 1997; Barnett 1998; Shipman 1999; Shipman et al. 2000; Shackleton et al. 2000; Shipman et al. 2000; Ebner et al. 2001; Smith et al. 2001; Smith et al. 2001a; Smith et al. 2001b; Barnett 2001; Yu and Miller 2001; Yu and Miller 2002; Toussaint and Igel 2002). The general idea behind most of this work is that the evolvability of a population, which is defined as the ability of random variations to sometimes produce improvements, is increased by the use of redundant representations. Furthermore, because redundant representations allow a population to change the genotype without changing the phenotype, the ability of a population to adapt after changes and the performance of GEAs should increase.

However, in most of this work the focus has not been on the performance of GEAs, but on characteristics of the search like reachability of phenotypes, evolvability of populations, or connectivity of search spaces. No results have been presented up till now that clearly indicate the superiority of redundant representations and neutral search on practical test problems or real-world instances. Recently, Knowles and Watson (2002) presented an investigation into the performance of neutral search for NK landscapes, H-IFF, and MAX-SAT problems. The results showed that using arbitrary redundant representations (random boolean network mapping) does not increase the performance of

mutation-based search for the considered test problems. In most of the problems used, adding redundancy appeared to reduce performance.

Although, at the moment, the focus in investigating the role of redundant representations is mainly on neutral mutations and their effects on search characteristics, there is other work which tries to address the effects of redundancy on the performance of evolutionary search. Researchers used different types of redundant representations and sometimes observed either an increase or a decrease in the performance of GEAs. Over time, different opinions regarding the effects of redundancy on the performance of GEAs have been developed. Some work noticed that redundant representations lead to a reduction in GEA performance (Davis 1989; Eshelman and Schaffer 1991; Ronald et al. 1995). The low performance was argued to be either due to a loss of diversity in the population, or because different genotypes that represent the same phenotype compete against each other. Also, the larger size of the search space was listed as a reason for lower GEA performance.

In contrast, other mostly application-oriented work reports higher performance with additional redundancy (Cohoon et al. 1988; Gerrits and Hogeweg 1991; Julstrom 1999), which some researchers ascribe to an increase in diversity that hinders premature convergence. Further work considered the computational implications of genetic code-like representations in gene expression (Kargupta 2000b; Kargupta 2001). Kargupta investigated how redundant representations influence the energy of the Fourier spectrum. The results show that using redundant representations and encoding phenotypes with higher fitness by a larger number of genotypes results in a higher energy of the Fourier spectrum, reduces the difficulty of the optimization problem, and therefore allows a more effective evolutionary search.

This short literature review has shown that the influence of redundant representations on the performance of GEAs is a strongly disputed topic. It can be expected that there is no easy and general answer, and not all types of redundant representations will be useful (Harvey and Thompson 1997). To find answers, it is necessary to characterize the different types of redundant representations regarding their specific properties, and to develop quantitative models describing how solution quality and run duration of GEAs is influenced. This approach can help to clear up some of the disputed questions and to find out under which circumstances which type of redundant representation can be beneficial for GEAs. Consequently, in the following section we develop a classification for different types of redundant representations and in Sects. 3.1.4 and 3.1.5 we develop quantitative models.

3.1.2 Synonymously and Non-Synonymously Redundant Representations

We give some basic definitions and develop a classification for different types of redundant representations which is based on their synonymity.

As we have discussed in Sect. 2.1.2 we have to distinguish between genotypes and phenotypes. Φ_g is defined as the genotypic search space, where the operators crossover and mutation are applied. Φ_p is the phenotypic search space. The fitness of an individual depends on the properties of the phenotype $x^p \in \Phi_p$. A representation $f_g : \Phi_g \rightarrow \Phi_p$ determines which phenotypes $x^p \in \Phi_p$ are represented by which genotypes $x^g \in \Phi_g$. We want to assume that every phenotype x^p is assigned to at least one genotype x^g. Otherwise, if a phenotype x^p is not represented by some genotype x^g this solution can never be found by the used optimization algorithm.

A representation f_g is redundant if the size of the genotypic search space is larger than the size of the phenotypic search space, $|\Phi_g| > |\Phi_p|$. This means, there are more different genotypes than phenotypes. When using search spaces where not all possible phenotypes or genotypes are accessible by the used search method, a representation is redundant if the number of accessible phenotypes is smaller than the number of accessible genotypes. Therefore, in general a representation f_g is redundant if on average one accessible phenotype is represented by more than one genotype. Redundant representations are less efficient encodings, which use an additional number of genes, but do not increase the encoded information content. Therefore, a representation is redundant if l different phenotypes are assigned to m different genotypes where $m > l$. Although the larger number of possible genotypes would allow us to encode more individuals than there are phenotypes, some of the information that exists in the genotypes is not considered.

Distinguishing between Synonymously and Non-Synonymously Redundant Representations

To classify different types of redundant representations we want to measure how similar the genotypes are that are assigned to the same phenotype. A representation is defined to be *synonymously redundant* if the genotypes that are assigned to the same phenotype are similar to each other. Consequently, we denote a representation to be *non-synonymously redundant* if the genotypes that are assigned to the same phenotype are not similar to each other. Therefore, the synonymity of a representation depends on the metric that is defined on Φ_g and Φ_p. The metric defined on Φ_p depends on the properties of the considered problem and the metric defined on Φ_g depends on the used search operator. Depending on different operators and metrics used on Φ_g we get different synonymity of the representation f_g. In Fig. 3.1, we illustrate the differences between synonymous and non-synonymous redundancy. For this illustrative example we use the Euclidean distance between the individuals for indicating how similar different individuals are.

We want to formalize the classification into synonymously and non-synonymously redundant representations. In general, a redundant representation f_g assigns a phenotype x^p to a set of different genotypes $x^g \in \Phi_g^{x^p}$, where $\forall x^g \in \Phi_g^{x^p} : f_g(x^g) = x^p$. All genotypes x^g in the genotypic set $\Phi_g^{x^p}$ represent

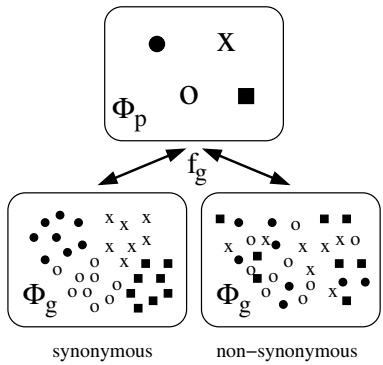


Figure 3.1. Synonymous versus non-synonymous redundancy. The different symbols indicate different genotypes and their corresponding phenotypes. When using synonymously redundant representations (left), genotypes that represent the same phenotype are similar to each other. When using non-synonymously redundant representations (right), genotypes that represent the same phenotype are not similar to each other but distributed over the whole search space

the same phenotype x^p. A representation is synonymously redundant if the genotypic distances between all $x^g \in \Phi_g^{x^p}$ are small for all different x^p. Therefore, if for all phenotypes the sum over the distances between all genotypes that correspond to the same phenotype

$$\left(\sum_{x^p} \frac{1}{2} \left(\sum_{x^g \in \Phi_g^{x^p}} \sum_{y^g \in \Phi_g^{x^p}} d(x^g, y^g) \right) \right), \tag{3.1}$$

where $x^g \neq y^g$, is reasonably small a representation is denoted to be synonymously redundant. $d(x^g, y^g)$ depends on the mutation operator used and measures the distance between two genotypes $x^g \in \Phi_g^{x^p}$ and $y^g \in \Phi_g^{x^p}$ which both represent the same phenotype x^p. The distance between two genotypes depends on their genotypic similarity and is small if the two genotypes are similar.

A different but equivalent approach of defining the synonymity of redundant representations is to use equivalence classes (compare the approach from Radcliffe outlined in Sect. 2.4.2). All genotypes x^g in the genotypic set $\Phi_g^{x^p}$ belong to the same equivalence class. If the sum of the genotypic distances between the individuals that belong to the same equivalence class is small, then the representation is synonymously redundant.

Synonymity and Locality of Redundant Representations

The synonymity of redundant representations is related to the locality of non-redundant representations (compare Sect. 3.3). A genotype x^g is a neighbor

to some other genotype y^g if the distance $d(x^g, y^g) = d_{min}$, where $d_{min} \neq 0$ is the minimal distance between two individuals in the genotypic search space. When using binary representations, $d_{min} = 1$ and two genotypes are neighbors if they differ in one allele. As discussed in Sect. 3.3, the locality of a representation describes how well neighboring genotypes correspond to neighboring phenotypes. If neighboring genotypes correspond to neighboring phenotypes, a representation has high locality and small changes in the genotype result in small changes in the corresponding phenotype. In contrast, representations have low locality if neighboring genotypes do not correspond to neighboring phenotypes. There is evidence, both analytical (for example Whitley (1999) for mutation-based search or Sect. 3.3.6 for crossover-based search), and empirical (for example Gottlieb et al. (2001) or Rothlauf and Goldberg (2000)), which shows for easy problems that low-locality representations result in low GEA performance. The genetic operators mutation and crossover no longer work properly as they create new offspring that are not similar to their parent(s).

Low-locality representations result in low GEA performance as guided search methods like GEAs, that use knowledge gained during search for determining future search steps, can only perform better than random search if on average similar solutions have similar fitness (individuals are similar if there are only a few search steps between them). In general, guided search methods assume that in the neighborhood of high-quality solutions other high-quality solutions can be found (Manderick et al. 1991; Horn 1995; Deb et al. 1997; Christensen and Oppacher 2001). High-quality solutions are grouped together and are not scattered over the whole search space (Radcliffe 1991a; Whitley 2002). Therefore, to perform well, guided search methods have to search more often in the neighborhood of already found promising high-quality solutions than around low-quality solutions (compare the optimal allocation of trials for the two-armed bandit problem as discussed in Holland (1975)). This behavior guarantees high performance of the search method if on average neighboring solutions have similar properties, which means the fitness values of neighboring solutions are correlated.

However, search heuristics could not use any information learned during the search for determining future search steps, and consequently show low performance if the fitness values of neighboring, or similar, genotypes are not correlated (Weinberger 1990; Manderick et al. 1991; Jones and Forrest 1995). On the one hand, the fitness values of neighboring genotypes can be uncorrelated if the problem itself is difficult, which means the fitness values of neighboring phenotypes are uncorrelated. Then, guided search methods will behave as a random search even in the presence of high-locality representations. On the other hand, the fitness values of neighboring genotypes are also uncorrelated if low-locality representations are used and neighboring genotypes do not correspond to neighboring phenotypes. The low-locality representations destroy existing correlations between phenotypes and their corresponding fitness values, and genotypic neighbors no longer have similar properties and fitnesses.

In this case, search heuristics can not use any information learned during the search for determining future search steps. As a result, it makes no sense for guided search approaches to search around already found high-quality genotypes and guided mutation-based search algorithms become random search. A mutation does not result in a solution with similar properties but in a random solution. Analogously, crossover is not able to create new solutions with similar properties to their parents, but creates new, random solutions. Therefore, high locality representations are a necessity for efficient evolutionary search. When using low-locality representations, no guided search is possible and guided search methods become random search.

These concepts can also be applied to redundant representations. When using non-synonymously redundant representations, genetic operators like mutation or crossover can result in an offspring that is phenotypically completely different from its parent(s). Therefore, non-synonymously redundant representations have the same effect on GEAs as when using low-locality representations. Using neutral search spaces where the connectivity between the phenotypes is strongly increased by the use of a redundant representation allows us to reach many different phenotypes in one single search step. However, increasing the connectivity between the phenotypes by using non-synonymously redundant representations results in random search and decreases the efficiency of GEAs. As for low-locality representations, a search step does not result in a similar phenotype but creates a randomly chosen individual. Therefore, guided search is no longer possible and guided search methods become random search. As a result, we get reduced GEA performance on problems that are easy for guided search methods (that means the fitness values of similar phenotypes are correlated) when using non-synonymously redundant representations. Examples for non-synonymously redundant representations are the direct binary mapping, the cellular automaton mapping, or the random boolean network mapping, which have been proposed by Shackleton et al. (2000). Although the use of these types of representations strongly increases the connectivity between phenotypes, we get low GEA performance as neighboring genotypes do not correspond to neighboring phenotypes. Initial evidence of the low performance of mutation-based search when using such non-synonymously redundant representations was shown by Knowles and Watson (2002) for the random boolean network mapping.

In contrast, when using synonymously redundant representations, the connectivity between the phenotypes is not increased. Therefore, small genotypic variations can not result in large phenotypic changes but either in the same, or a similar, phenotype. Figure 3.2 illustrates this behavior and compares it to non-synonymously redundant representations. Examples for synonymously redundant representations are the trivial voting mapping (Shackleton et al. 2000) which is investigated more closely in Sect. 3.1.6.

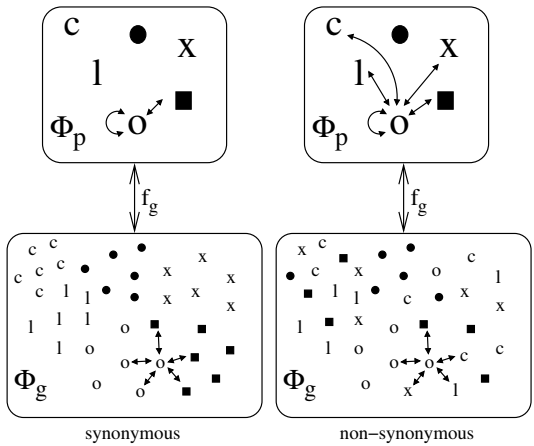


Figure 3.2. The effects of small mutation steps for synonymously versus non-synonymously redundant representations. The different symbols indicate different genotypes and their corresponding phenotypes. The arrows indicate search steps which result in neighboring individuals. When using synonymously redundant representations, a mutation results in either the same or a similar phenotype. In contrast, when using non-synonymously redundant representations the mutation of a genotype results in completely different phenotypes.

Formalizing Synonymously Redundant Representations

In this subsection, we introduce some quantities that can be used for characterizing the properties of synonymously redundant representations. We use the definitions from Sects. 2.1.2 and 3.1.2.

To describe a redundant representation, we introduce k_r, the *order of redundancy*. k_r is defined as $\log(|\Phi_g|)/\log(|\Phi_p|)$ and measures the amount of redundant information in the encoding. There are k_r bits and 2^{k_r} different possibilities (individuals) to encode 1 Bit of information. When using binary genotypes and binary phenotypes, the order of redundancy can be calculated as

$$k_r = \frac{\log(2^{l_g})}{\log(2^{l_p})},$$

where l_g is the length of the binary genotype and l_p is the length of the binary phenotype. When using a non-redundant representation, the number of genotypes equals the number of phenotypes and $k_r = 1$.

Furthermore, we want to characterize not only to what degree a representation is redundant, but also in what way it is redundant. We are especially interested in the overrepresentation and underrepresentation of specific solutions. Therefore, we introduce r as the number of genotypes that represent the one phenotype that has the highest fitness. When using non-redundant representations, every phenotype is assigned to exactly one genotype and $r = 1$. However, in general, $1 \leq r \leq |\Phi_g| - |\Phi_p| + 1$.

In the following discussion, we want to focus on how redundant representations influence the behavior of selectorecombinative GAs. Selectorecombinative GAs use crossover as the main search operator and mutation only serves as a background operator. When focusing on selectorecombinative GAs we implicitly assume that there are building blocks (BBs) and that the GA process schemata. Consequently, we must define how k_r and r depends on the properties of the BBs.

In general, when looking at BBs of size k there are 2^k different phenotypic BBs which are represented by 2^{kk_r} different genotypic BBs. Therefore,

$$k_r = \frac{k_g}{k_p},$$

where k_g denotes the genotypic size of a BB and k_p the size of the corresponding phenotypic BB. As before, a representation is redundant if $k_r > 1$. The size of the genotypic BBs is k_r times larger than the size of the phenotypic BB. Furthermore, r is defined as the number of genotypic BBs of length kk_r that represent the best phenotypic BB of size k. Therefore, in general,

$$r \in \{1, 2, \ldots, 2^{kk_r} - 2^k + 1\}. \tag{3.2}$$

In contrast to k_r, which is determined by the representation used, r depends not only on the representation used, but also on the specific problem that should be solved. Different instances of a problem result in different values of r. If we assume that k_r is an integer (each phenotypic allele is represented by k_r genotypic alleles) the possible values of the number of genotypic BBs that represent the optimal phenotypic BB can be calculated as

$$r = i^k, \text{ with } i \in \{1, 2, \ldots, 2^{k_r} - 1\}. \tag{3.3}$$

A representation is *uniformly redundant* if all phenotypes are represented by the same number of different genotypes. Therefore, when using a uniformly redundant representation every phenotypic BB of size $k = k_p$ is represented by

$$r = 2^{k(k_r - 1)} \tag{3.4}$$

different genotypic BBs. Table 3.1 gives an example for a uniformly redundant encoding. Two bits in a phenotype x^p are represented by four bits in the genotype x^g. Therefore, $k_r = 2$ and $r = 4$. With $|\Phi_p| = 2^k = 2^2$ the size of the genotypic space is $|\Phi_g| = 2^{kk_r} = 2^4 = 16$.

x^g	x^p
00 00, 00 01, 01 00, 01 01	0 0
10 00, 10 01, 11 00, 11 01	1 0
00 10, 01 11, 00 11, 01 11	0 1
10 10, 10 11, 11 10, 11 11	1 1

Table 3.1. An example of a uniformly redundant representation, where $k_r = 2$ and $r = 4$

By introducing redundancy the search space for a GA using binary phenotypes of string length $l = l_p$ is increased from $|\Phi_p| = 2^l$ to $|\Phi_g| = 2^{lk_r}$. The length of the individuals increases from $l = l_p$ in the phenotypic space to $l_g = k_r \times l$ in the genotypic space. To represent all phenotypes, each individual $x^p \in \Phi_p$ must be represented by at least one genotype $x^g \in \Phi_g$. If $|\Phi_g| = |\Phi_p|$, and each phenotype is represented by at least one genotype, we have a non-redundant, one-to-one mapping.

3.1.3 Complexity Model for Redundant Representations

For modeling the effects of redundant representations on the performance of GEAs, we can use the complexity model from Goldberg (1991a) and Goldberg et al. (1992). If we want to understand the effects of redundancy, we must decompose the problem into smaller sub-problems and try to solve these separately. We could decompose the problem step by step and subsequently collate all the sub-problems. The decomposition (Goldberg 1998) takes place as follows:

- GAs process building blocks.
- Problems are tractable by BBs.
- GAs must ensure proper supply of BBs in the initial generation.
- GAs must grow the high quality BBs.
- GAs must mix the BBs well.
- GAs must decide well among competing BBs.

This decomposition gives us a framework for investigating the effects of redundancy on the performance of GAs. We want to examine how redundancy affects the problem decomposition point by point:

Using redundant encodings does not change the principal behavior of GAs. After adding redundancy, GAs still process building blocks.

A problem is still tractable by building blocks when using a redundant encoding. However, the question arises as to whether the size of building blocks is changed by redundant encodings. On one hand, redundant representations increase the number of bits that are part of a building block in the genotype from k to kk_r. On the other hand, the number of building blocks in the genotype that represent the same BB in the phenotype increases from 1 to on average $2^{k(k_r-1)}$. Furthermore, the number of different fitness values that can be assigned to the genotypes remains constant. Taking these effects into account, we assume that problems are still tractable by BBs and the larger size of the genotypic BBs is compensated by the higher number of genotypic BBs.

How do redundant encodings change the initial supply of BBs in the initial population? For uniform redundancy (every phenotype $x^p \in \Phi_p$ is represented by the same number of genotypes $x^g \in \Phi_g$), the initial supply of BBs $x_0/N = 1/2^k$ is the same as for non-redundant representations. If the number of genotypes x^g that represent a phenotype x^p is above average, then

the phenotype x^p, and the containing schemata h^p, are overrepresented in the initial population. GEAs are pushed more towards solutions that are similar to these x^p. Analogously, the performance of GEAs decreases if the proportion of less fit phenotypes in the initial population is increased by redundancy. As a result, the supply of BBs could be modified by redundant encodings.

For a proper growth and mixing of BBs, above average BBs must be preferred by selection, and BBs should not be disrupted by the crossover operator. As the genotypic defining length of a BB $\delta(h^g)$ increases when using redundant representations, GEA operators that do not obey the linkage disrupt BBs more frequently. To overcome this problem, competent GAs (Mühlenbein and Paaß 1996; Goldberg 1999; Larranaga et al. 1999; Mühlenbein and Mahnig 1999; Pelikan 2002) could be used. These kind of GEAs obey the linkage, and genotypic building blocks h^g are not disrupted by recombination. Thus, no reduction of performance should occur, and redundancy should have no negative effect on the proper mixing of BBs. However, when using redundant representations there are different genotypes x^g and y^g that represent the same phenotype x^p. Recombining x^g and y^g could result in offspring that do not represent x^p. This effect is also known as cross-competition among isomorphic identical BBs. As an example for a one-bit problem, the genotype $\{00\}$ represents the phenotype $x^p = 0$, and the genotypes $\{01, 10, 11\}$ represent the phenotype $y^p = 1$. If the genotypes 01 and 10 are recombined, the possibilities for the offspring are 00 and 11. Although both parental genotypes represent the same phenotype y^p, one of the offspring represents x^p. Cross-competition among isomorphic identical BBs is a result of using non-synonymously redundant representations, which were discussed in the previous subsection. Non-synonymously redundancy randomizes genetic search as new BBs are introduced into the search that did not exist in the parents. When using such representations, the recombination of genotypes that represent the same phenotype can result in completely different new genotypes and phenotypes (compare also Fig. 3.2). When using synonymously redundant representations, cross-competition can not occur.

Finally, the decision making between competing BBs is not affected by using redundant representations. With redundancy there are different genotypic BBs h^g that represent the same phenotypic BB h^p, but the selection process does not decide between the different h^g because they all have the same fitness. The fitness evaluation is based on the fitness of the phenotypic BBs h^p, and not their genotypic representation.

After recognizing that redundancy could have a major effect on the supply, and a minor effect on the proper mixing of building blocks, we want to quantify its effect on BBs supply in the following subsection.

3.1.4 Population Sizing for Synonymously Redundant Representations

In Sect. 3.1.2, we described how non-synonymously redundant representations result in randomized search as they increase the connectivity of the search space. Therefore, in this section we want to focus on synonymously redundant representations and develop a population sizing model that describes their influence on the performance of selectorecombinative GAs.

As we focus in our investigation on selectorecombinative GAs we can use the existing theory describing the behavior of selectorecombinative GAs from Harik et al. (1997) and Thierens and Goldberg (1994). They describe for non-redundant representations how the population size and the time to convergence that is necessary to solve a specific problem depend on the characteristics of the problem.

Following Harik et al. (1997) the probability that a GA with a population size N converges after t_{conv} generations to the correct solution is

$$P_n = \frac{1-(q/p)^{x_0}}{1-(q/p)^N},$$

where x_0 is the expected number of copies of the best BB in the randomly initialized population, $q = 1-p$, and p is the probability of making the right choice between a single sample of each BB

$$p = \mathbb{N}\left(\frac{d}{\sqrt{2m'}\sigma_{BB}}\right). \tag{3.5}$$

$\mathbb{N}$ is the cumulative distribution function for a normal distribution, d is the signal difference between the best BB and its strongest competitor, $m' = m-1$ with m is the number of BBs in the problem, σ^2_{BB} is the variance of a BB, and $q = 1-p$ is the probability of making the wrong decision between two competing BBs. It has been shown in Harik et al. (1997) that this random walk or Gambler's ruin model can be used for describing the behavior of selectorecombinative GAs propagating schemata and BBs. In the following paragraphs, this model is the basis for describing the influence of synonymously redundant representations on the behavior of GAs.

For a randomly initialized population with no redundancy, $x_0 = N/2^k$. The situation changes when using redundant representations. Then, the initial supply depends on the characteristics of the representation, namely r and k_r. With r the number of genotypic BBs of length kk_r that represent the best phenotypic BB of length k, we get

$$x_0 = N\frac{r}{2^{kk_r}}, \tag{3.6}$$

where k_r is the order of redundancy. The assumption that redundant representations affect the initial supply of BBs is the core idea behind the proposed

model describing the influence of synonymously redundant representations on GA performance. We assume that other effects of synonymously redundant representations on GA performance can be neglected. Consequently, when using uniformly redundant representations, $r = 2^{k(k_r-1)}$ and $x_0 = N/2^k$. These are the same values as when using non-redundant representations. Therefore, GA performance does not change when using uniformly redundant representations.

As the variance σ_{BB}^2 and the number m of BBs is not affected by the use of a redundant representation, the probability of GA failure $\alpha = 1 - P_n$ can be calculated as

$$\alpha = 1 - \frac{1 - (q/p)^{x_0}}{1 - (q/p)^N}. \tag{3.7}$$

If we assume that x_0 is small and $q < p$ we can assume that $1 - (q/p)^N$ converges to 1 faster than $1 - (q/p)^{x_0}$. Using these approximations (see also Harik et al. (1997)) the equation can be simplified to

$$\alpha \approx \left(\frac{1-p}{p}\right)^{x_0}.$$

Therefore, for the population size we get

$$N \approx \frac{2^{kk_r}}{r}\left(\frac{\ln(\alpha)}{\ln\left(\frac{1-p}{p}\right)}\right). \tag{3.8}$$

The normal distribution in (3.5) can be approximated using the first two terms of the power series expansion (see Abramowitz and Stegun (1972)) as $\mathbb{N}(x) \approx 1/2 + x/2$, where $x = d/\sqrt{\pi m'}\sigma_{BB}$. Substituting p from (3.5) into (3.8) we get:

$$N \approx \frac{2^{kk_r}}{r}\ln(\alpha)/\ln\left(\frac{1-x}{1+x}\right),$$

Since x is a small number, $\ln(1-x)$ can be approximated with $-x$ and $\ln(1+x)$ with x. Using these approximations we finally get for the population size N:

$$N \approx -\frac{2^{k_r k-1}}{r}\ln(\alpha)\frac{\sigma_{BB}\sqrt{\pi m'}}{d}. \tag{3.9}$$

The population size N goes with $O\left(\frac{2^{k_r}}{r}\right)$ when using synonymously redundant representations. With increasing r the number of individuals that are necessary to solve a problem decreases. Using a uniformly redundant representation, where $r = 2^{k(k_r-1)}$, does not change the population size N in comparison to non-redundant representations.

3.1.5 Run Duration and Overall Problem Complexity for Synonymously Redundant Representations

To describe the performance of GAs, we must calculate not only the number of individuals that are necessary for solving a problem, but also the expected number of generations until convergence.

Based on Mühlenbein and Schlierkamp-Voosen (1993) and Thierens and Goldberg (1994), Miller and Goldberg developed a convergence model for selectorecombinative GAs (Miller and Goldberg 1996b; Miller and Goldberg 1996a). The convergence time t_{conv} depends on the length of the phenotypes $l = l_p$ and the used selection scheme. Using the selection intensity I the convergence model is

$$p(t) = 0.5\left(1 + \sin\left(\frac{It}{\sqrt{l}} + \arcsin(2p(0) - 1)\right)\right),$$

where $p(0) = x_0/N$ is the proportion of best building blocks in the initial population. I depends only on the used selection scheme. The number of generations t_{conv} it takes to fully converge the population can be calculated by putting $p(t_{conv}) = 1$:

$$t_{conv} = \frac{\sqrt{l}}{I}\left(\frac{\pi}{2} - \arcsin\left(2p(0) - 1\right)\right). \tag{3.10}$$

If we assume $k = 1$ and uniform redundancy (equal proportion of 1s and 0s in the initial population) we get $p(0) = 0.5$. Then, the number of generations until convergence simplifies to

$$t_{conv} = \frac{\pi}{2}\frac{\sqrt{l}}{I}.$$

With redundancy the initial proportion of building blocks is $p(0) = \frac{r}{2^{kk_r}}$ (see (3.6)). Using $\arcsin(x) = x + o(x^3)$ the time until convergence could be approximated by

$$t_{conv} \approx \frac{\sqrt{l}}{I}\left(1 + \frac{\pi}{2} - \frac{r}{2^{k_r k-1}}\right). \tag{3.11}$$

With increasing $r/2^{k_r}$ the time to convergence t_{conv} is reduced. Therefore, the optimal solution is found after a lower number of generations if it is overrepresented by the synonymously redundant representation. For uniform redundancy $r = 2^{k(k_r-1)}$, we get

$$t_{conv} \approx \frac{\sqrt{l}}{I}\left(1 + \frac{\pi}{2} - \frac{1}{2^{k-1}}\right).$$

The time until convergence when using uniformly redundant representations is the same as without redundancy.

After we have calculated the number of individuals that are necessary for solving a problem (see (3.9)), and the number of generations that GAs using only crossover need to converge (see (3.11)), we can calculate the absolute number of fitness calls that are necessary for solving a problem:

$$\begin{aligned} N \times t_{conv} &\approx -\frac{2^{k_r k-1}}{r} \ln(\alpha) \frac{\sigma_{BB}\sqrt{\pi m'}}{d} \times \frac{\sqrt{l}}{I} \left(1 + \frac{\pi}{2} - \frac{r}{2^{k_r k-1}}\right) \\ &= \frac{\sqrt{\pi l m'}}{I} \ln(\alpha) \frac{\sigma_{BB}}{d} \left(1 - \frac{2^{k k_r}}{4r}(2+\pi)\right) \end{aligned}$$

The overall number of fitness calls goes with $O(2^{k_r}/r)$. In comparison to non-redundant representations, the number of fitness calls stays constant for synonymously redundant representations if $r = 2^{k(k_r-1)}$. Then $x_0/N = 1/2^k$ and the representation is uniformly redundant.

3.1.6 Analyzing the Redundant Trivial Voting Mapping

In the previous subsection, we developed theoretical models describing how synonymously redundant representations influence the quality of the solution and the time that is necessary to find the good solutions. In this subsection, we investigate whether or not the proposed models allow a good prediction of GA performance for the trivial voting (TV) mapping. The TV mapping is a synonymously redundant representation and we use it for one-max and concatenated deceptive trap problems. During our investigation we are particularly interested in whether the developed models allow us to accurately predict the expected solution quality and running time of a selectorecombinative GA.

The Trivial Voting Mapping

We give a short introduction into the trivial voting mapping.

When using the TV mapping, a set of mostly consecutive, genotypic alleles is relevant for the value of one allele in the phenotype. Each allele in the genotype can only influence the value of one allele in the phenotype. The value of the phenotypic allele is determined by the majority of the values in the genotypic alleles. In general, the different sets of alleles in the genotype defining one phenotypic allele have the same size. The TV mapping is a synonymously redundant representation as all genotypes that represent the same phenotype are similar to each other. A mutation in a genotype results either in the same corresponding phenotype, or in one of its neighbors.

The TV mapping can be easily characterized using the representation parameters defined in Sect. 3.1.2. The order of redundancy k_r is simply the number of genotypic alleles that determine the value of one phenotypic allele. Figure 3.3 gives an example for the TV mapping.

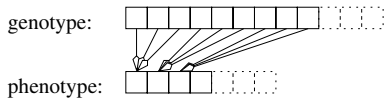


Figure 3.3. The trivial voting mapping

Shackleton et al. (2000) applied the TV mapping to binary strings in the context of the neutral theory. When used for binary strings, binary genotypes $x^g \in \mathbb{B}^{l_g}$ are assigned to binary phenotypes $x^p \in \mathbb{B}^{l_p}$. The length of a genotype is larger than the length of a phenotype, $l_g > l_p$. The value of one phenotypic bit is determined by the majority of the values in the corresponding genotypic bits (majority vote). However, if k_r is even then the number of ones could equal the number of zeros. Therefore, half the cases that result in a tie should encode a one in the corresponding phenotypic allele, and half the cases should represent a zero. For example, for $k_r = 4$ the genotypic BBs 1100, 1010, and 1001 represent a 1 and the genotypic BBs 0011, 0101, 0110 represent a zero.

Because the majority of the votes determines the values of the corresponding phenotypic allele, the TV mapping is a uniformly redundant representation. Each phenotypic BB is represented by the same number of genotypic BBs which is $2^{k(k_r-1)}$, where k is the size of the phenotypic BB.

As we are not only interested in uniformly redundant representations, but also want to know how non-uniformly redundant representations influence GA performance, we extend the TV mapping to allow the encoding to overrepresent some individuals. Therefore, we want to assume that if the number of ones in the genotypic alleles $x^g_{k_r i+j}$, where $i \in \{0, \ldots, l_p - 1\}$ and $j \in \{0, \ldots, k_r - 1\}$, is larger or equal than a constant u then the value of the phenotypic allele x^p_i is set to one. Vice versa, the phenotypic allele x^p_i is set to zero if less than u of the corresponding genotypic alleles are set to one. Therefore,

$$x^p_i = \begin{cases} 0 \text{ if } & \sum_{j=0}^{k_r-1} x^g_{k_r i+j} < u \\ 1 \text{ if } & \sum_{j=0}^{k_r-1} x^g_{k_r i+j} \geq u, \end{cases}$$

where $u \in \{1, \ldots, k_r\}$. x^g_i (respectively x^p_i) denotes the ith allele of the genotype (respectively phenotype). u can be interpreted as the number of genotypic alleles that must be set to one to encode a one in the corresponding phenotypic allele. This representation is denoted as *extended trivial voting* (eTV) mapping. For $u = (k_r + 1)/2$ (k_r must be odd) we get the original TV mapping. Extending the TV mapping in the proposed way allows us to investigate how non-uniform redundancy influences the performance of GAs.

When using the eTV mapping, the number r of genotypic BBs that can represent the optimal phenotypic BB depends on the number of ones in the genotypic alleles that determine the value of the corresponding phenotypic allele. Considering (3.3) we get

$$r = \left(\sum_{j=u}^{k_r} \binom{k_r}{j} \right)^k, \tag{3.12}$$

where $u \in \{1, \ldots, k_r\}$. We assume that a BB is optimal if all phenotypic bits are set to $x_i^p = 1$. k denotes the size of the phenotypic BB. To give a short illustration, we use a redundant representation with $k_r = 3$, $k = 1$ (compare Fig. 3.3). The optimal BB is $x_i^p = 1$. Because $u \in \{1, \ldots, k_r\}$ there are three different values possible for r. For $u = 1$ the phenotypic allele x_i^p is set to one if at least one of the three corresponding genotypic alleles $x_{ik_r}^g$, $x_{ik_r+1}^g$, or $x_{ik_r+2}^g$ is set to one. Therefore, a one in the phenotype is represented by $r = \sum_{j=1}^{3} \binom{k_r}{j} = 7$ different genotypic BBs (111, 110, 101, 011, 100, 010, and 001). For $u = 2$, the optimal genotypic BB $x_i^p = 1$ is represented by $r = \sum_{j=2}^{3} \binom{k_r}{j} = 4$ different genotypic BBs (111, 110, 101, and 011) and the representation is uniformly redundant. For $u = 2$ we get the original TV mapping. For $u = 3$, the optimal phenotypic BB is represented only by one genotypic BB (111).

Experiments and Empirical Results

Here we present empirical results when using the binary trivial voting mapping for the one-max problem and the concatenated deceptive trap problem.

One-Max Problem

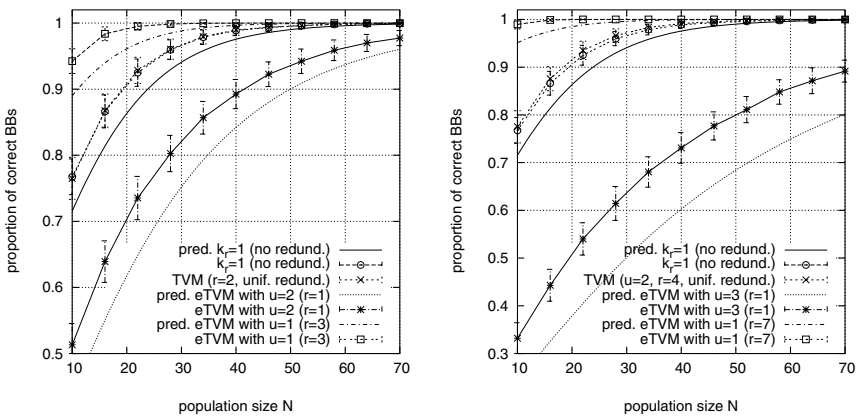
The first test example for our empirical investigation is the one-max problem. This problem is very easy to solve for GEAs as the fitness of an individual is simply the number of ones in the binary phenotype. To ensure that recombination results in a proper mixing of the BBs, we use uniform crossover for all experiments with the one-max problem. Furthermore, in all runs we use tournament selection without replacement and a tournament size of 2. For the one-max function the signal difference d equals 1, the size k of the building blocks is 1, and the variance of a building block $\sigma_{BB}^2 = 0.25$.

x_i^p	$x_{2i}^g x_{2i+1}^g$ (with $k_r = 2$)		
	extended TV		original TV
	$r = 1$	$r = 3$	$r = 2$
0	00, 01, 10	00	00, 01
1	11	01, 10, 11	10, 11

Table 3.2. The trivial voting mapping for $k_r = 2$

When using the binary TV mapping for the one-max problem each bit of a phenotype $x^p \in \Phi_p$ is represented by k_r bits of the genotype $x^g \in \Phi_g$. The string length of a genotype x^g is $l_g = k_r \times l_p$ and the size of the genotypic search space is $|\Phi_g| = 2^{k_r l_p}$. Table 3.2 illustrates for $k_r = 2$ the two possibilities ($r = 1$

and $r = 3$) of assigning genotypic BBs $\{00, 01, 10, 11\}$ to one of the phenotypic BBs $\{0, 1\}$ when using the extended TV mapping described in the previous paragraphs. With denoting x_i^p the value of the ith bit in the phenotype, the $2i$th and $(2i+1)$th bit of a genotype determine x_i^p. Because the size of the BBs $k = 1$, the number of genotypic BBs that represent the optimal phenotypic BB is either $r = 1$ or $r = 3$ (compare (3.12)). Furthermore, Table 3.2 also lists the case where $r = 2$. This case is the original uniformly redundant TV mapping. The second bit of each genotypic BB does not contribute to the construction of the phenotype.



(a) $k_r = 2$ (b) $k_r = 3$

Figure 3.4. Experimental and theoretical results of the proportion of correct BBs on a 150-bit one-max problem using the trivial voting mapping for $k_r = 2$ (left) and $k_r = 3$ (right). The lines without line points show the theoretical predictions. When using non-uniformly redundant representations, GA performance is changed with respect to the overrepresentation or underrepresentation of the high-quality BBs.

		extended TV mapping			original TV
		$u = 1$	$u = 2$	$u = 3$	mapping
$k_r = 2$	r	3	1	-	2
	x_0/N	3/4	1/4	-	$2/4 = 1/2$
$k_r = 3$	r	7	4	1	4
	x_0/N	7/8	$4/8 = 1/2$	1/8	$2/4 = 1/2$

Table 3.3. Properties of the different TV mappings for the one-max problem ($k = 1$)

In Fig. 3.4(a) ($k_r = 2$) and Fig. 3.4(b) ($k_r = 3$), the proportion of correct BBs at the end of a run for a 150 bit one-max problem using the TV mapping is shown. For this problem 2^{150} different phenotypes are represented by either

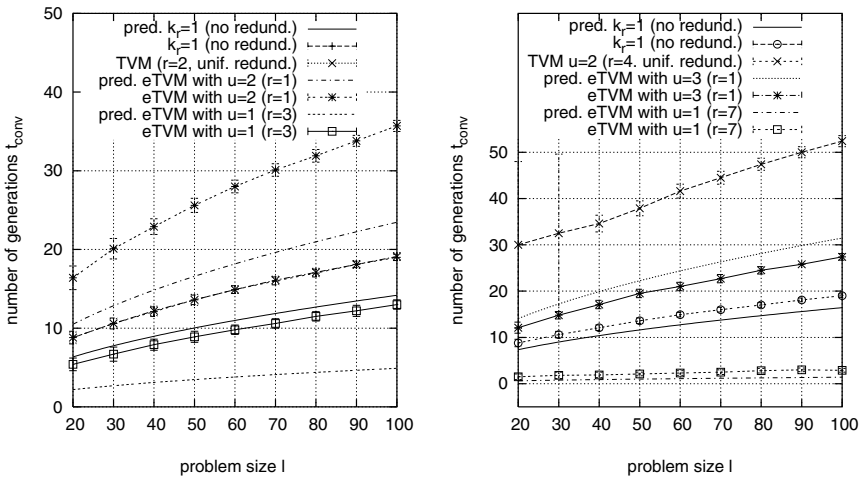
2^{300} ($k_r = 2$) or 2^{450} ($k_r = 3$) different genotypes. If we use the eTV mapping (indicated in the plots as eTVM) we can set u either to 1 or 2 ($k_r = 2$) or to 1, 2, or 3 ($k_r = 3$). The corresponding values for r, which can be calculated according to (3.12), as well as x_0/N are shown in Table 3.3. x_0 is the expected number of copies of the best BB in the initial population and N is the population size. Furthermore, the figures show the results when using the original, uniformly redundant TV mapping, and when using the non-redundant representation with $k_r = 1$. The lines without line points show the theoretical predictions from (3.7), and the lines with line points show the empirical results which are averaged over 250 runs. The error bars indicate the standard deviation.

The results show that for the uniformly redundant TV mapping, $r = 2$ ($k_r = 2$) or $r = 4$ ($k_r = 3$), we get the same performance as for using the non-redundant representation ($k_r = 1$). As in the original model proposed by Harik et al. (1997) the theoretical model slightly underestimates GA performance. As predicted by our model which we proposed in Sect. 3.1.4, GA performance does not change when using a uniformly redundant representation. Furthermore, we can see that if the optimal BB is underrepresented ($u = 2$ for $k_r = 2$ and $u = 3$ for $k_r = 3$) GA performance decreases. Equation 3.7 gives us a good prediction for the expected solution quality if we consider that the non-uniform redundancy of the representation changes the initial BB supply according to (3.6). If the optimal solution is overrepresented ($u = 1$ for both cases, $k_r = 2$ and $k_r = 3$) GA performance increases. Again the theoretical models give a good prediction for the expected proportion of correct BBs.

Summarizing the results, we can see that using the uniformly redundant TV mapping does not change GA performance as compared to using the non-redundant representation. Only if we overrepresent the optimal phenotypic BB, does GA performance increase; likewise, if we underrepresent the optimal BB, GA performance drops. As our derived model is able to make accurate predictions for the expected solution quality, our assumption that synonymously redundant representations influence GA performance by changing the initial supply seems to be valid.

In the remaining paragraphs, we perform an empirical investigation into the effect of the TV mapping on the number of generations until the population of a selectorecombinative GA converges. Again we use the one-max problem and the TV mapping from above with the same parameters except the population size is set to $N = 2l_p$ to allow reliable decision making for the one-max problem (Goldberg et al. 1992). As we use tournament selection without replacement of size two the selection intensity $I = 1/\sqrt{\pi}$.

Figures 3.5(a) ($k_r = 2$) and 3.5(b) ($k_r = 3$) show the number of generations that are necessary until 90% of all phenotypic BBs are found over the problem size which is equal to $l = l_p$. The lines without line points show the predictions from (3.10) and the lines with line points plot the empirical results. We can see that the run duration of a GA when using the non-redundant representa-



(a) $k_r = 2$ (b) $k_r = 3$

Figure 3.5. Theoretical predictions and experimental results for the number of generations that are necessary until 90% of all phenotypic BBs are correctly identified. The plots are for one-max problems and trivial voting mapping with $k_r = 2$ (left) and $k_r = 3$ (right).

tion ($k_r = 1$) is exactly the same as when using the uniformly redundant TV mapping with $k_r = 2$. For $k_r = 3$ and $u = 2$ (uniform redundancy) the run duration is slightly increased in comparison to the non-redundant encoding. We expect that this difference increases with larger k_r. In agreement with the results from Thierens (1995), we report a small underestimation of the expected number of generations when using either non-redundant, or uniformly redundant, representations.

When using non-uniformly redundant variants of the eTV mapping the underestimation is larger, but nevertheless the model gives a good approximation for the expected number of generations. We can see that increasing r increases the run duration. For example, if each phenotypic bit is represented by three genotypic bits ($k_r = 3$) and a one is represented if at least one out of three genotypic bits is set to one ($u = 1$) then a GA finds the good solutions after very short time (compare eTVM with $u = 1$). The expected number of generations shows the predicted behavior. The necessary number of generations increases by about $O(\sqrt{l})$. We see that the proposed model allows us to make good predictions for the expected run duration.

Concatenated Deceptive Trap Problem

Our second test example uses deceptive trap functions. Traps were first used by Ackley (1987) and investigations into the deceptive character of these func-

tions were provided by Deb and Goldberg (1993). Figure 3.6 depicts a 3-bit deceptive trap problem where the size of a BB is $k = 3$. The fitness value of a phenotype x^p depends on the number of ones u in the string of length l. The best BB is a string of l ones which has fitness l. Standard GEAs are misled to the deceptive attractor which has fitness $l - 1$. For the 3-bit deceptive trap the signal difference d is 1, and the fitness variance equals $\sigma^2_{BB} = 0.75$. We construct a test problem for our investigation by concatenating $m = 10$ of the 3-bit traps so we get a 30-bit problem. The fitness of an individual x is calculated as $f(x) = \sum_{i=0}^{m-1} f_i(u)$, where $f_i(u)$ is the fitness of the ith 3-bit trap function from Fig. 3.6. Although this function is difficult for GEAs it can be solved with proper population size N.

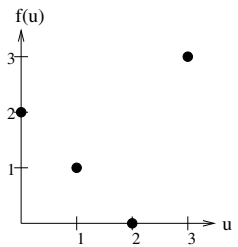


Figure 3.6. A 3-bit deceptive trap problem

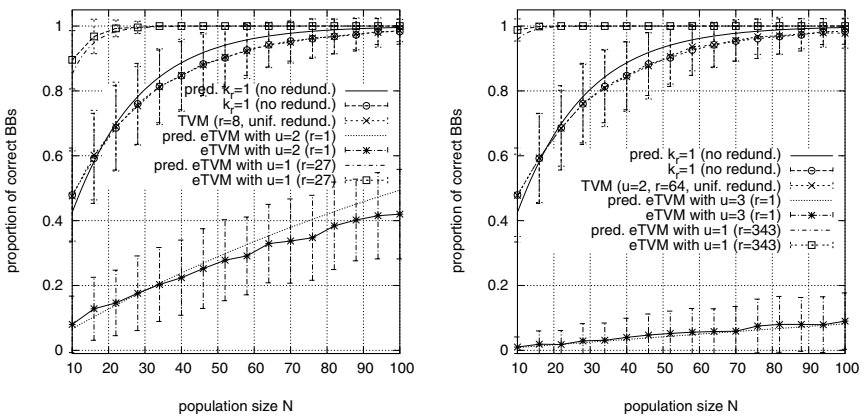
For deceptive traps of size $k = 3$ we can calculate the number r of genotypic BBs that represent the optimal phenotypic BB according to (3.12). Table 3.4 summarizes, for the binary TV mapping, how r and x_0/N depends on u, which describes how many of the genotypic alleles must be set to 1 to encode a 1 in the phenotype. x_0 is the expected number of copies of the best BB in the initial population and N is the population size. We have also included the properties of the original uniformly redundant TV mapping.

		extended TV mapping			original TV
		$u = 1$	$u = 2$	$u = 3$	mapping
k_r=2	r	$3^3 = 27$	$1^3 = 1$	-	$2^3 = 8$
	x_0/N	27/64	1/64	-	8/64 = 1/8
k_r=3	r	$7^3 = 343$	$4^3 = 64$	$1^3 = 1$	$4^3 = 64$
	x_0/N	343/512	64/512 = 1/8	1/512	64/512 = 1/8

Table 3.4. Properties of the different TV mappings for the trap of size $k = 3$

By analogy to the previous paragraphs, Figs. 3.7(a) ($k_r = 2$) and 3.7(b) ($k_r = 3$) show the proportion of correct BBs at the end of a run over different population sizes for ten concatenated 3-bit deceptive trap problems. In this problem, 2^{30} different phenotypes are represented by either 2^{60} ($k_r = 2$) or 2^{90} ($k_r = 3$) different genotypes. As before, we use tournament selection without replacement of size 2. In contrast to the one-max problem, two-point crossover was chosen for recombination. Uniform crossover would result in an improper

mixing of the BBs because the genotypic BBs are either of length $l_g = k_r l_p = 6$ ($k_r = 2$), or of length $l_g = 9$ ($k_r = 3$). Again, the lines without line points show the predictions of the proposed model for different r. Furthermore, empirical results, which are averaged over 250 runs, are shown for various values of r. The results show that for the uniformly redundant TV mapping we get the same performance as when using the non-redundant representation ($k_r = 1$). As in the experiments for the one-max problem the proposed model predicts the experimental results well if the eTV mapping is used and some BBs are underrepresented or overrepresented.



(a) $k_r = 2$ (b) $k_r = 3$

Figure 3.7. Experimental and theoretical results of the proportion of correct BBs for ten concatenated 3-bit deceptive traps. We show results for different variants of the TV mapping and $k_r = 2$ (left) and $k_r = 3$ (right). The lines without line points show the theoretical predictions. As predicted, GA performance sharply decreases if the eTV mapping underrepresents the optimal BB.

The presented results show that the effects of synonymously redundant representations like the TV mapping on the performance of GEAs can be explained well by a change of the initial supply of high-quality BBs. If the eTV mapping favors high-quality BBs then the performance of GAs is increased. If good BBs are underrepresented the performance is reduced. If the representation is uniformly redundant, GAs show the same performance as when using the non-redundant encoding.

3.1.7 Conclusions and Further Research

This section investigated how redundant representations influence the performance of GEAs. It distinguished between synonymously and non-synonymous-

ly redundant representations and illustrated that non-synonymous redundancy does not allow genetic operators to work properly and therefore reduces the efficiency of evolutionary search. When using synonymously redundant representations, GEA performance depends on the change of the initial supply. Based on this observation, models were developed that give the necessary population size for solving a problem, and the number of generations as $O(2^{k_r}/r)$, where k_r is the order of redundancy and r is the number of genotypic BBs that represent the optimal phenotypic BB. As a result, uniformly redundant representations do not change the behavior of GAs. Only by increasing r, which means overrepresenting the optimal solution, does GA performance increase. In contrast, GA performance decreases if the optimal solution is underrepresented. Therefore, non-uniformly redundant representations can only be used advantageously if information exists a-priori regarding the optimal solution. The validity of the proposed theoretical concepts is illustrated for different variants of the redundant trivial voting mapping. The results show that the developed population sizing and time to convergence models allow an accurate prediction of the expected solution quality and solution time.

The proposed classification, population sizing, and time to convergence models allow us to evaluate redundant representations in a systematic and theory-guided matter. This approach will help users and researchers to answer some of the disputed questions regarding the benefits of redundant representations and to use redundant representations such that they increase the performance, reliability and efficiency of evolutionary computation methods.

In this study, we only considered crossover-based search and neglected the influence of redundant representations on mutation-based search. However, we believe that many of the discussed topics are also relevant when using mutation. According to Sect. 3.1.2, we believe that in analogy to the results from Knowles and Watson (2002), using non-synonymously redundant representations reduces the performance of mutation-based search. As these representations have low locality, mutation will not work properly and the search becomes random. Furthermore, there is some theoretical evidence (Radcliffe 1991a; Whitley et al. 1997; Rana and Whitley 1997; Whitley 1999; Whitley 2000a; Christensen and Oppacher 2001) that mutation-based search only performs well if the connectivity of the phenotypic search space is preserved by the used representation. If the connectivity is either not preserved, such as for low locality representations, or greatly increased (which results in a reduction of the relevant connectivity) like in many non-synonymously redundant representations, the performance of mutation-based search decreases. In contrast, we expect when using synonymously redundant representations that mutation-based search will show similar behavior and performance as when using crossover-based search. Using synonymously redundant representations introduces many plateaus in the fitness landscape but does not change the structure of the search space. Mutation can still easily find neighboring phenotypes. When using non-uniformly redundant representations, some plateaus in the fitness landscape have a larger size which increases the probability

that mutation finds the solution represented by the genotypes forming this plateau. As a result, the performance of mutation-based search increases if a synonymously redundant representation overrepresents the optimal solution, and decreases otherwise.

3.2 Scaling

This section provides the second of three elements of a theory of representations and addresses representations which change the importance of alleles when mapping genotypes on phenotypes. Common representations for GEAs often encode the phenotypes by using a sequence of alleles. When assigning phenotypes to genotypes, a representation can change the importance of the alleles. For example, a phenotype is a list of integers and all alleles (integers) are equally relevant for calculating the fitness of a phenotype. We know from previous work that the BBs (alleles) are solved in parallel if all alleles are equally relevant (Goldberg 1989c). The situation that all alleles are equally relevant is equivalent to the situation that the BBs are uniformly scaled. However, when encoding the phenotypic integers using binary strings, the contributions of the genotypic bits to the fitness function are no longer equal and some bits are more relevant than others. When using such non-uniformly scaled representations, the BBs are solved sequentially and domino convergence occurs. Therefore, the time to convergence increases and the genetic search is affected by genetic drift. This means that lower salient BBs are fixed before they can be reached by the search process.

Based on previous work (Rudnick 1992; Thierens 1995; Thierens et al. 1998; Harik et al. 1997), we describe how the performance of GEAs is influenced by the use of representations with non-uniformly scaled BBs. We develop a population-sizing model with, and without, considering genetic drift. The theoretical models are verified with empirical results.

In the following subsection, we review the effects of domino convergence and genetic drift. In Sects. 3.2.2 and 3.2.3 we develop population sizing models for domino convergence with, and without, considering drift. We present empirical verification of the proposed models in Sect. 3.2.4 and end with concluding remarks.

3.2.1 Definitions and Background

Representations assign phenotypes x^p consisting of different alleles x_i^p, where $i \in \{0, \ldots, l_p\}$, to genotypes, which also consist of different alleles x_i^g, where $i \in \{0, \ldots, l_g\}$. Furthermore, the function f_p assigns to each phenotype x^p a corresponding fitness value. Usually, the fitness of a phenotype depends on the values of x_i^p. Therefore, different phenotypic alleles can have a different contribution to the fitness of an individual x^p. Representations are defined to be *uniformly scaled* if the genotype-phenotype mapping f_g does not change

the contribution of the alleles to the calculation of the fitness function. For example, when all phenotypic alleles x_i^p have an equal contribution to the fitness of an individual, representations are uniformly scaled if the importance of the alleles does not change by the genotype-phenotype mapping, and all genotypic alleles x_i^g also have an equal fitness contribution. Representations can also be uniformly scaled if different phenotypic alleles have a different contribution to the fitness function as the scaling of a representation only considers the genotype-phenotype mapping and not the phenotype-fitness mapping.

Representations are defined to be *non-uniformly scaled* if the contributions of the alleles to the fitness are different for the genotypes than the phenotypes. The most frequently used non-uniformly scaled representations are *exponentially scaled* representations where the genotypic alleles have an exponentially scaled contribution to the values of the phenotypic alleles. A common example for exponentially scaled representations is the binary encoding, which encodes integer phenotypic alleles using binary genotypes. The contributions of the genotypic alleles to the construction of the phenotypic alleles are exponentially different.

We want to emphasize that the scaling of a representation only considers the genotype-phenotype mapping and is independent of the phenotype-fitness mapping. For the discussion in this section, we assume that the phenotypic alleles are uniformly scaled. This means that all phenotypic alleles have the same contribution to the fitness of an individual.

The term "domino convergence" was introduced by Rudnick (1992). He defined domino convergence as the sequential convergence of the alleles in a bitstring. Domino convergence occurs if the alleles are non-uniformly scaled and solved sequentially. Rudnick showed that there is a *convergence window* of size λ_c. The convergence window is a set of λ_c contiguous alleles that have started to converge but are not yet fully converged. More salient alleles have already converged completely, whereas lower salient alleles are not yet touched by convergence. The existence of the convergence window means that not all parts of a problem are solved at the same speed. The higher the contribution of one allele to the overall fitness of an individual, the earlier this allele is solved by GEAs.

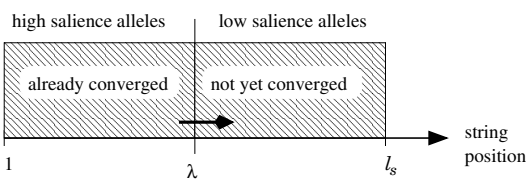


Figure 3.8. Domino convergence

The λ model is a formal approach for modeling domino convergence in GEAs (Thierens et al. 1998). $\lambda \in \{1, \ldots, l_s\}$ defines the dividing line between the genotypic alleles that have already converged and those that are still not

touched by the selection pressure of GEAs. l_s is the genotypic length of an exponentially scaled BB. We want to assume that each phenotypic BB of size l_p is encoded using an (exponentially scaled) genotypic BB of size l_s. λ moves on from $\lambda = 1$, where all alleles are assumed to be in the initial random state, to $\lambda = l_s$ where all alleles (of the genotypic BB) are converged (see Fig. 3.8). At the beginning of the genetic search process (low λ) alleles are solved that have a high contribution to the fitness of an individual, whereas at the end of the GEA search ($\lambda \approx l_s$) alleles with a low influence on the fitness are solved. For intermediate states, all lower salient alleles have never been exposed to direct selection pressure and remain in their initial state as long as there is no genetic drift. In the approach used by Thierens et al. (1998), the convergence window has size $\lambda_c = 1$.

If the population size N is not large enough, some of the lower salient genotypic alleles are randomly fixed due to genetic drift. This results in a degradation of GEAs. The existence of genetic drift is widely known and has been addressed in the field of population genetics (Kimura 1962; Kimura 1964; Gale 1990; Nagylaki 1992; Hartl and Clark 1997), and also in the field of genetic algorithms (Goldberg and Segrest 1987; Asoh and Mühlenbein 1994; Thierens et al. 1998; Lobo et al. 2000). More information regarding the influence of genetic drift on the performance of GEAs can be found in Sect. 3.1.1. Lower salient bits drift randomly because selection does not take these bits into account when deciding between solutions. As two absorbing states exist for binary alleles (all individuals have at their ith position either a zero or a one), lower salient bits could be fixed because of random genetic drift before they are directly exposed to selection pressure and solved by GEAs.

Thierens (1995) developed a convergence time model for non-uniformly scaled problems. This work modeled the domino convergence for tournament selection by a wave equation, and showed that the overall convergence time complexity for an exponentially scaled fitness function is approximately of order $O(l)$. This is much slower than for uniformly scaled problems where ranked based selection mechanisms have a convergence time of order $O(\sqrt{l})$ where l is the length of the bitstring.

In the following paragraphs, we want to use some existing models for developing a population sizing model for exponentially scaled representations, with, and without considering the effects of genetic drift.

3.2.2 Population Sizing Model for Exponentially Scaled Representations Neglecting the Effect of Genetic Drift

Representations are exponentially scaled if different genotypic alleles have a different, exponentially scaled, contribution to the construction of the phenotypic alleles. The binary representation is the most common representative of exponentially scaled representations.

We use a genetic algorithm for solving a problem and assume domino convergence due to the exponentially scaled representation. Furthermore, there

should be no overlapping between the solving process of the current and the next allele ($\lambda_c = 1$). The next lower salient bit is only solved after the current allele is completely converged. Thus, we expect the developed models to only give us a lower bound on the solution quality and run duration of GAs. Although this is a very conservative assumption, this model (Thierens et al. 1998) predicts the behavior of a GA well (Lobo et al. 2000). If we want to overcome the limitation of strict sequential solving, a convergence window of larger size $\lambda_c > 1$ can be introduced. This would result in an overlapping solving process for the different alleles in the convergence window. The convergence window would move through the string from the most salient to the least salient alleles and would allow us to create a more exact model for domino convergence.

Assuming strictly sequential solving of alleles ($\lambda_c = 1$), the fitness variance of an exponentially scaled string when λ bits are converged during a GA run was calculated in Thierens et al. (1998) as

$$\sigma_N^2(\lambda) = \frac{x_0}{N}(1 - \frac{x_0}{N})\frac{2^{2(l_s-\lambda)} - 1}{3} \approx \frac{x_0}{N}(1 - \frac{x_0}{N})\frac{2^{2(l_s-\lambda)}}{3},$$

where l_s is the length of the exponentially scaled string, x_0 is the average number of best solutions in the initial population, and N is the population size. For $x_0 = N/2$ (we assume that l_s genotypic bits encode a phenotypic BB of size $k = 1$) the variance simplifies to $\sigma_N^2(\lambda) \approx \frac{1}{12}2^{2(l_s-\lambda)}$. This means that the fitness variance is determined only by the non-converged region. The more alleles that are solved during the run, the less noise we get. The fitness variance of the genotypic allele that is currently solved is

$$\sigma_{BB}^2(\lambda) = \frac{x_0}{N}(1 - \frac{x_0}{N})2^{2(l_s-\lambda)}.$$

For $x_0 = N/2$ we get $\sigma_{BB}^2 = 2^{2(l_s-\lambda-1)}$. As the contribution of the alleles to the fitness of an individual becomes less with lower salience, σ_{BB}^2 becomes smaller with increasing time. The fitness distance d between the best individual and its strongest competitor could be calculated as

$$d(\lambda) = 2^{l_s-\lambda}.$$

If we concatenate m exponentially scaled genotypic BBs of size l_s, we get competing BBs and an overall genotypic string length of $l_g = l = l_s m$. This means, we increase the number of phenotypic BBs from one to m and each phenotypic BB of size l_p is encoded by l_s genotypic alleles. When solving the λth bit of a genotypic BB there is noise σ_{BB}^2 from the λth bit of the competing $m' = m-1$, other BBs, and noise σ_N^2 from the yet unfixed bits in each BB. We know from past work that the probability of making the right choice between a single sample of each bit (compare also (3.5)) is (Miller 1997):

$$p = \mathbb{N}\left(\frac{d}{\sqrt{2(m'\sigma_{BB}^2 + m\sigma_N^2)}}\right).$$

Using the equations from above results in

$$p = \mathbb{N}\left(\frac{1}{\sqrt{2\frac{x_0}{N}\left(1-\frac{x_0}{N}\right)\left(\frac{4}{3}m-1\right)}}\right) \tag{3.13}$$

For $x_0 = N/2$ (we assume that each phenotypic BB has size $k = l_p = 1$ and each phenotypic BB is encoded using l_s genotypic alleles) we finally get:

$$p = \mathbb{N}\left(\sqrt{\frac{2}{\frac{4}{3}m-1}}\right). \tag{3.14}$$

The probability of deciding well is independent of the position λ in the string as long as there is no genetic drift, and the proportion of zeros and ones remains constant for the yet unfixed bits. This means for m exponentially scaled BBs that the probability p of deciding well is independent of the length l_s of the exponentially scaled BB and that it stays constant over all alleles. Thus, the proportion of correct bits in the string at the end of a run depends only on the number of BBs m. For $m = 1$ (there is only one exponentially scaled BB) there is no noise from competing BBs and we get $p = \mathbb{N}\left(\sqrt{6}\right)$. Using p we can calculate the proportion of incorrect bits at the end of the run according to the Gambler's ruin model (compare also (3.15)) as (Feller 1957; Harik et al. 1997)

$$\alpha = 1 - \frac{1-(1/p-1)^{x_0}}{1-(1/p-1)^{N}}. \tag{3.15}$$

For $x_0 = N/2$ and $z = \sqrt{\frac{2}{\frac{4}{3}m-1}}$ we get:

$$\alpha = 1 - \frac{1-\left(\frac{1}{\mathbb{N}(z)}-1\right)^{N/2}}{1-\left(\frac{1}{\mathbb{N}(z)}-1\right)^{N}} = \frac{\left(\frac{1}{\mathbb{N}(z)}-1\right)^{N/2}}{1+\left(\frac{1}{\mathbb{N}(z)}-1\right)^{N/2}} = \frac{1}{1+\left(\frac{1}{\mathbb{N}(z)}-1\right)^{-N/2}}. \tag{3.16}$$

The probability α of GA failure for exponentially scaled problems only depends on the population size N and the number of competing BBs m as long as the population size is large enough, and no genetic drift occurs. Notice that in contrast to the proportion of correct bits $1-\alpha$, the number of correctly found exponentially scaled BBs is $(1-\alpha)^{l_s}$ (each of the m genotypic BB has l_s alleles). When using the first two terms of the power series expansion as an approximation for the normal distribution (Abramowitz and Stegun 1972) from (3.14) we get

$$p = \frac{1}{2} + \frac{1}{\sqrt{2\pi}}z,$$

where $z = \sqrt{\frac{2}{\frac{4}{3}m-1}}$. Substituting this approximation in (3.15) results in

$$N = 2\ln\left(\frac{\alpha}{1-\alpha}\right) / \ln\left(\frac{1-\sqrt{\frac{2}{\pi}}z}{1+\sqrt{\frac{2}{\pi}}z}\right).$$

Since z tends to be a small number, $\ln(1 \pm z\sqrt{\frac{2}{\pi}})$ may be approximated as $\pm z\sqrt{\frac{2}{\pi}}$. Using these approximations and substituting the value z into the equation finally gives

$$N \approx -\frac{1}{2}\ln\left(\frac{\alpha}{1-\alpha}\right)\sqrt{\pi\left(\frac{4}{3}m-1\right)}. \tag{3.17}$$

This rough approximation determines more clearly the variables the population size N depends on. We see that for exponentially scaled representations, the necessary population size N grows with the square root of the size m of the problem. In contrast to the more general population sizing equation from Harik et al. (1999), N does not depend on the distance d and the variance of an allele σ_{BB} if genetic drift is neglected. As already mentioned, we want to emphasize that we consider an exponentially scaled representation (genotype-phenotype) mapping and do not address the phenotype-fitness mapping. For the developed model we assume that the size of the phenotypic BB, $k_p = 1$, and that the phenotypic alleles are uniformly scaled this means that all phenotypic alleles have the same contribution to the fitness of an individual.

Finally, we give an estimation for the convergence time t_{conv} for exponentially scaled BBs of length l_s. The time until a uniformly scaled string of length m is converged can be calculated (Thierens and Goldberg 1994) as

$$t = \frac{\pi}{2}\frac{\sqrt{m}}{I},$$

where I denotes the selection intensity. For tournament selection without replacement of size 2, $I = 1/\sqrt{\pi}$. As there are m exponentially scaled BBs, and therefore m alleles of the same salience, the GEAs solve m bits in parallel. The next m lower salient bits are solved when all m currently solved bits are fully converged ($\lambda_c = 1$). Thus, the solving process for exponentially scaled problems is strictly serial and goes with $O(l_s)$ (Thierens 1995, pp. 66ff). The overall time to convergence can be calculated as:

$$t_{conv} = l_s\frac{\pi}{2}\frac{\sqrt{m}}{I} = \frac{l}{\sqrt{m}}\frac{\pi}{2I}, \tag{3.18}$$

where $l = l_g = l_s m$. In contrast to an uniformly scaled representation ($t_{conv} = O(\sqrt{l_s m})$), the time to convergence goes with $O(l_s\sqrt{m})$ when using a representation, that assigns to each of the m phenotypic BBs l_s exponentially scaled alleles. We see clearly that GEAs need more time to converge if non-uniformly scaled representations are used.

3.2.3 Population Sizing Model for Exponentially Scaled Representations Considering the Effect of Genetic Drift

In the previous subsection, we have developed a population sizing and convergence time model for exponentially scaled representations. However, the model does not consider the effects of genetic drift. This subsection lifts this restriction and investigates the modifications necessary for considering genetic drift.

Drift affects genetic search if the drift time is lower than the time to convergence, $t_{drift} < t_{conv}$. Low salient bits are fixed due to drift before they can be reached by the solution process. The drift time has been studied in the context of GEAs (Goldberg and Segrest 1987; Asoh and Mühlenbein 1994). These studies show that the expected time for an allele to converge due to genetic drift is proportional to the population size N. For an equal proportion of ones and zeros in the start population, using tournament selection of size $s = 2$, the size of BBs $k = 1$ (Lobo et al. 2000), and random sampling with replacement, we get for the drift time

$$t_{drift} \approx 1.4N. \tag{3.19}$$

For $t_{conv} > t_{drift}$ genetic drift fixes some low salient bits before they can be solved by the search process. Using (3.18), we can calculate the population size N_{drift} for which GEAs using tournament selection of size 2 (selection intensity $I = 1/\sqrt{\pi}$) are affected by genetic drift as

$$N_{drift} < \frac{5\pi}{14}\sqrt{\frac{\pi}{m}}l\cdot \tag{3.20}$$

If the population size N of GEAs using an exponentially scaled representation is lower than N_{drift} then domino convergence does not reach the lowest salient alleles and these alleles are randomly fixed at one of the two absorbing states 0 or 1 (we assume a binary encoding with $\chi = 2$).

In the following paragraphs, we want to propose two approaches modeling the influence of drift on the performance of GEAs using exponentially scaled representations. At first, we need a model that describes the drift process itself. An approximation for the probability $s(t)$ that an allele is fully converged due to genetic drift at time (generation) t was given by Kimura (1964):

$$s(t) \approx 1 - 6\frac{x_0}{N}(1 - \frac{x_0}{N})\exp(-t/N)\cdot$$

Using this approximation we can calculate how the probability of randomly fixing an allele depends on the population size N and the number of generations t. When $x_0 = N/2$ is the expected number of 1s in the randomly initialized population (we still assume that all phenotypic BBs have size $k = 1$ and each phenotypic BB is represented by l_s genotypic bits) we get

$$s(t) = 1 - \frac{3}{2}\exp(-t/N)\cdot$$

As $s(t)$ is only an approximation ($s(t) < 0$ for small t) we define the convergence probability

$$s'(t) = \begin{cases} 0 & \text{for } t < -N\ln(2/3), \\ 1 - \frac{3}{2}\exp(-t/N) & \text{for } t > -N\ln(2/3). \end{cases}$$

For $t < -N\ln(2/3)$ the probability that an allele converges due to genetic drift is zero. In Fig. 3.9, we plot the probability that an allele is fully converged at generation t for $N = 10$.

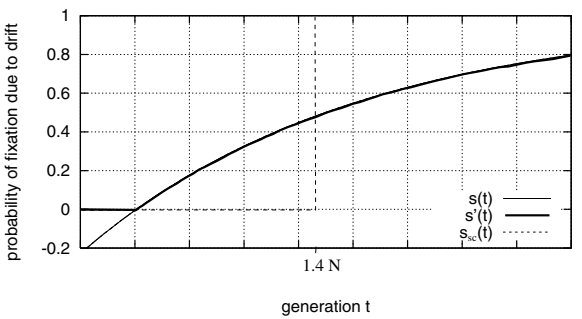


Figure 3.9. Genetic drift models

We know from (3.18) that GEAs need $t = \frac{\pi}{2}\frac{\sqrt{m}}{I}$ generations for solving one allele in the bitstring. When considering genetic drift, the bits in the string are either solved correctly with probability $1 - \alpha$ or they are randomly fixed due to genetic drift. If domino convergence reaches after $t = \lambda\frac{\pi}{2}\frac{\sqrt{m}}{I}$ generations an allele, this allele is not converged due to genetic drift with probability $1-s'(t)$. Therefore, we want to assume that with probability $1-s'(t)$, this allele remains in its initial state and is solved correctly with probability $1 - \alpha$. In contrast, with probability $s'(t)$, this allele is converged due to genetic drift. If the allele is converged due to genetic drift, it converges to the correct solution with probability 1/2. Therefore, using the probability of GA failure α from (3.16) we can calculate the probability of GA failure when considering genetic drift as

$$\alpha_{drift}(\lambda) = \left(1 - s'(\lambda\frac{\pi}{2}\sqrt{\pi m})\right)\alpha + \frac{1}{2}s'(\lambda\frac{\pi}{2}\sqrt{\pi m}),$$

when using tournament selection of size 2 ($I = 1/\sqrt{\pi}$). $1 - s'(\lambda\frac{\pi}{2}\sqrt{\pi m})$ is the probability that the allele at the λth position is not converged due to genetic drift at time $t = \lambda\frac{\pi}{2}\sqrt{\pi m}$. The probability of error for this allele is α. Furthermore, the alleles are affected by genetic drift with probability $s'(\lambda\frac{\pi}{2}\sqrt{\pi m})$, and converge to the wrong solution with probability 1/2. Consequently, the overall average percentage $\bar{\alpha}_{drift}$ of incorrect alleles in one genotypic BB of length l_s can be calculated as

$$\bar{\alpha}_{drift} = \frac{1}{l_s}\sum_{\lambda=1}^{l_s}\alpha_{drift}(\lambda) \tag{3.21}$$

and is simply the sum over all l_s bits of one BB. We refer to this model as the *approximated drift model*.

We assume in this model that the proportion of ones and zeros for for the unconverged alleles x_i^g, where $i > \lambda$, is still 0.5, and therefore α does not depend on λ. We also assume that the solving time for one allele stays constant over the whole solving process, and that it is independent of λ.

In the remaining paragraphs, we want to develop a second population sizing model for exponentially scaled alleles considering the effects of genetic drift. Instead of using the approximation from Kimura (1964), we use a simple drift/no-drift approach. We assume that all instances of an allele are either fixed due to genetic drift, or remain in the initial state. Thus, we get a linear, stair-case model for the estimation of α_{drift}, if we assume that for $t_{drift} > \lambda\frac{\pi}{2}\sqrt{\pi m}$ all bits can be solved with probability $1-\alpha$, and for $t_{drift} < \lambda\frac{\pi}{2}\sqrt{\pi m}$ the remaining low salient $l_s - \lambda$ bits of each BB are fixed randomly at the correct solution with probability 0.5. As previously, the drift time can be calculated for GAs using tournament selection of size $s = 2$ as $t_{drift} \approx 1.4N$. Therefore, the probability s_{sc} that an allele is fully converged due to genetic drift at time t is

$$s_{sc}(t) = \begin{cases} 0 & \text{for } t < t_{drift}, \\ 1 & \text{for } t > t_{drift}. \end{cases}$$

We illustrate this in Fig. 3.9. For $t < t_{drift}$ we assume no genetic drift, and for $t > t_{drift}$ all the remaining bits are randomly fixed. Therefore, the probability of GA failure can be calculated as:

$$\alpha'_{drift}(\lambda) = \begin{cases} \alpha & \text{for } \lambda < \frac{2.8N}{\pi\sqrt{\pi m}}, \\ 0.5 & \text{for } \lambda \geq \frac{2.8N}{\pi\sqrt{\pi m}}. \end{cases}$$

By using (3.19) and (3.20) the average percentage of incorrect alleles is:

$$\bar{\alpha}'_{drift}(\lambda) = \begin{cases} \frac{1}{l_s}\left(\sum_{\lambda=0}^{\lfloor\frac{2.8N}{\pi\sqrt{\pi m}}\rfloor}\alpha + \sum_{\lambda=\lceil\frac{2.8N}{\pi\sqrt{\pi m}}\rceil}^{l_s-1}\frac{1}{2}\right) & \text{for } N < \frac{5\pi}{14}\sqrt{\pi m}l_s, \\ \alpha & \text{for } N \geq \frac{5\pi}{14}\sqrt{\pi m}l_s. \end{cases} \tag{3.22}$$

For large N, no genetic drift occurs and we get the same failure probability as for the non-drift case (3.16). For small N the most salient $\lfloor\frac{2.8N}{\pi\sqrt{\pi m}}\rfloor$ bits are solved correctly with probability $1 - \alpha$ and the rest of the alleles are fixed randomly due to genetic drift.

As the drift time has a standard deviation of approximately the same order (Gale 1990, pp. 82ff.) as its mean ($\approx 1.4N$), the model underestimates the solution quality for $t < 1.4N$, and overestimates it for $t > 1.4N$. The probability of converging to the correct solution has a stair-cased slope regarding N as long as $t_{drift} < t_{conv}$. Thus, we refer to this model as the *stair-case drift model*.

3.2.4 Empirical Results for BinInt Problems

In this subsection, we illustrate that the proposed models considering genetic drift predict the behavior of GEAs well for exponentially scaled representations and small populations. We show that with decreasing number of competing exponentially scaled BBs, and increasing length of the BBs, genetic drift leads to a stronger decline of GEAs.

For our empirical investigation we use the integer one-max problem from Sect. 5.1. Furthermore, we encode the phenotypes (integers) as binary strings using the binary encoding (compare Sect. 5.2). As a result we get the BinInt problem (Rudnick 1992). There are l_s genotypic bits that encode the phenotype and the contribution of an allele to the construction of the phenotype x^p is exponentially scaled. The fitness of a phenotype is its integer value. Therefore, the overall fitness of an individual can be calculated as:

$$f(x^g) = \sum_{i=0}^{l_s-1} x_i^g 2^i$$

Thus, the optimal solution is a string with only ones. If the population size is large enough, the BinInt problem is easily solved by GEAs in a step-wise way according to the domino convergence model.

According to Sect. 2.1.2, the BinInt problem can be separated into a representation f_g, and a phenotype-fitness mapping f_p. Therefore, the genotype is a binary string of length l_s and the phenotype is an integer. We assume that the phenotype-fitness mapping is the identity function and assigns an integer value to each phenotype,

$$f_p(x^p) = x^p.$$

The representation f_g is exponentially scaled and assigns l_s genotypic alleles to the phenotype. The value of a phenotype is calculated as

$$x^p = \sum_{i=0}^{l_s-1} x_i^g 2^i.$$

We present no results for exponentially scaled representations and deceptive problems because the population size N that is necessary to solve these types of problems is in general large enough to ensure that no genetic drift occurs, and therefore, the available population sizing models from Sect. 3.2.2 can be used.

For all experiments we use uniform crossover, no mutation, and tournament selection of size two without replacement. The initial population is generated randomly (the initial state of the population has an equal proportion of zeros and ones, $x_0 = N/2$) and a GA run is stopped after the population is fully converged. To gain statistical evidence we performed 250 runs for each problem. We present results for three test cases:

- one BinInt problem ($m = 1$),
- 10 concatenated BinInt problems ($m = 10$),
- 50 concatenated BinInt problems ($m = 50$).

We want to denote the correct solution (a sequence of l_s ones) for one BinInt problem as a BB. For the BinInt problem, the size of the phenotypic BBs is $k = 1$ and the size of the genotypic BBs is l_s. The fitness of an individual is the sum over all m BinInt problems and can be calculated as

$$f(x^p) = \sum_{i=0}^{m-1} x_i^p = \sum_{i=0}^{m-1} \sum_{j=0}^{l_s-1} x^g_{l_s i+j}.$$

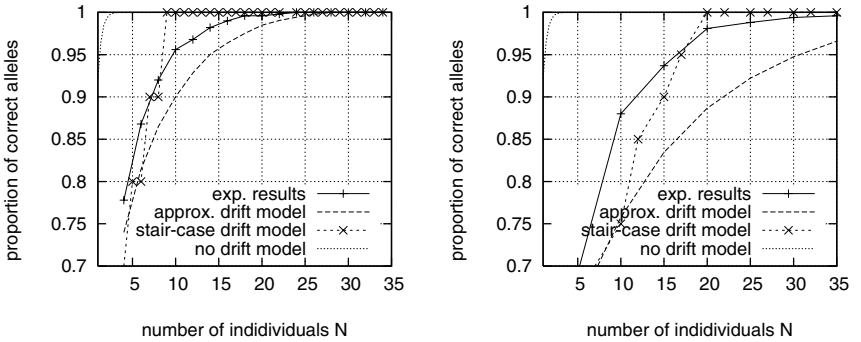
For each of these three test cases we present results for $l_s = 5$ and $l_s = 10$. Therefore, the maximum fitness of a BinInt problem is $f(x^p) = 2^5 = 32$ ($l_s = 5$) or $f(x^p) = 2^{10} = 1,024$ ($l_s = 10$). In Table 3.5, we present the overall string length l_g, the probability p of making the right choice between a single sample of each BB, and the overall convergence time t_{conv} when assuming no genetic drift. If drift occurs some lower salient genotypic alleles are randomly fixed at 0 or 1 before they can be reached by the search process and t_{conv} is an upper bound for the overall convergence time.

Table 3.5. Some properties of the three test cases

	$m = 1$		$m = 10$		$m = 50$	
l_s	5	10	5	10	5	10
l_g	5	10	50	100	250	500
p	$\mathbb{N}(\sqrt{6})$		$\mathbb{N}(\sqrt{6/37})$		$\mathbb{N}(\sqrt{6/197})$	
t_{conv}	$5\frac{\pi}{2}\sqrt{\pi}$	$10\frac{\pi}{2}\sqrt{\pi}$	$5\frac{\pi}{2}\sqrt{10\pi}$	$10\frac{\pi}{2}\sqrt{10\pi}$	$5\frac{\pi}{2}\sqrt{50\pi}$	$10\frac{\pi}{2}\sqrt{50\pi}$

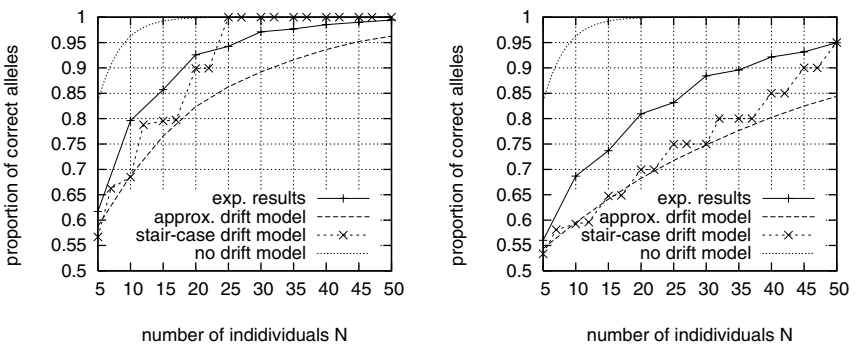
In Figs. 3.10 ($m = 1$), 3.11 ($m = 10$), and 3.12 ($m = 50$), we present results for the different test cases. One phenotypic integer is represented by either $l_s = 5$ (left) or $l_s = 10$ (right) bits. The solid lines with line points show the empirical results. We show predictions for considering no drift (dotted line), for the stair-case drift model (dashed line with line points), and for the approximated drift model (dashed line). All predictions consider domino convergence.

For all three cases, genetic drift has a large impact on the GA performance especially with increasing l_s, and decreasing number of competing BBs m. The number of competing BBs is equivalent to the number of concatenated BinInt problems. In contrast to the no-drift model which could not predict the behavior of the GA well, both the stair-case drift model, as well as the approximated drift model, are able to accurately describe the behavior of GEAs using exponentially scaled representations. Both models consider genetic drift and predict the behavior of GEAs better than the domino-convergence model alone.



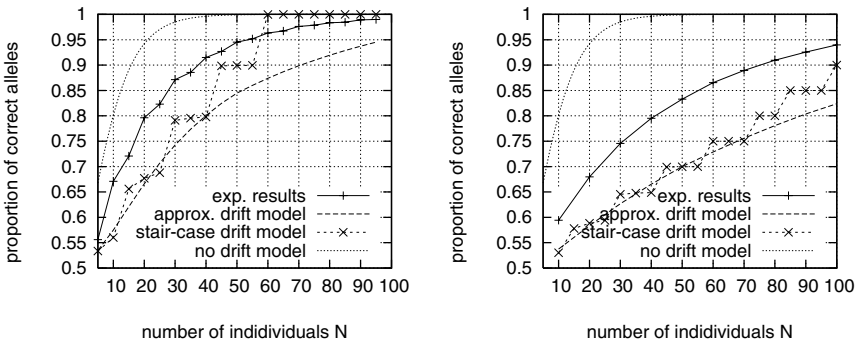
(a) $m = 1$, $l_s = 5$ (b) $m = 1$, $l_s = 10$

Figure 3.10. Experimental and theoretical results of the proportion of correct alleles for one BinInt problem of length $l_s = 5$ (left) and $l_s = 10$ (right). The solid lines with line points show the empirical results. All shown predictions consider domino convergence. We show predictions for considering no drift (dotted line), for the stair-case drift model (dashed line with line points), and for the approximated drift model (dashed line). With increasing length l_s of the genotype, genetic drift has a major impact on the GA, and the GA performance declines.



(a) $m = 10$, $l_s = 5$ (b) $m = 10$, $l_s = 10$

Figure 3.11. Experimental and theoretical results of the proportion of correct alleles for $m = 10$ concatenated BinInt problems of length $l_s = 5$ (left) and $l_s = 10$ (right). The overall string length is $l = 50$ (left) and $l = 100$ (right). The solid lines with line points show the empirical results. We show predictions for considering no drift (dotted line), for the stair-case drift model (dashed line with line points), and for the approximated drift model (dashed line). Although the GA is affected by noise from the competing $m - 1$ problems, which superimposes the effects of drift, genetic drift has a major impact on the GA performance with increasing l_s.



(a) $m = 50$, $l_s = 5$ (b) $m = 50$, $l_s = 10$

Figure 3.12. Experimental and theoretical results of the proportion of correct alleles for $m = 50$ BinInt problems of length $l_s = 5$ (left) and $l_s = 10$ (right). The overall string length is $l = 250$ (left) and $l = 500$ (right). The solid lines with line points show the empirical results. We show predictions for considering no drift (dotted line), for the stair-case drift model (dashed line with line points), and for the approximated drift model (dashed line). With increasing l_s, genetic drift has a larger impact on the GA, and the GA performance declines.

We see that with increasing l_s, the performance of GEAs declines. This behavior is expected as we know from our theoretical investigations that t_{conv} increases linearly with l_s (see (3.18)), whereas the drift time t_{drift} stays constant (see (3.19)). Therefore, with increasing l_s more and more lower salient alleles are fixed due to genetic drift and GEA performance declines.

Our results show that the influence of genetic drift is reduced for an increasing number m of BBs (BinInt problems). We know from (3.14) that with increasing m the probability of making the right choice between a single sample of each bit is reduced. Therefore, larger populations N are necessary which reduce the influence of genetic drift with increasing m. This relationship can be seen nicely in the presented plots. For $m = 1$ (see Fig. 3.10) we have no competing BBs, the necessary populations are very small, and the GA is strongly affected by genetic drift. Therefore, the no-drift model fails completely. For $m = 50$, however, there is a lot of noise from the competing BBs (BinInt problems), and therefore larger populations are necessary. The influence of genetic drift is smaller, and the no-drift model when only considering domino convergence gives an acceptable prediction of the solution quality (see. Fig. 3.12).

As predicted, the stair-case model underestimates the proportion of correct alleles for small N ($N < \frac{5}{7}t_{drift}$), and overestimates it for large populations ($N > \frac{5}{7}t_{drift}$). The approximated drift model predicts the slope of the empirical results well, but due to the used domino convergence with strict sequential solving of the bits, it always underestimates the proportion of correct alleles. We believe that by introducing a convergence window, and assuming some

parallel solving of the alleles, that the approximated drift model should more accurately predict the behavior of GAs.

We have seen that the lower the number m of competing BBs (BinInt problems) is, and the more bits l_s each BinInt problem has, the stronger is the impact of genetic drift on the solution quality. Although we only use a strictly serial solution process and no convergence window, the developed models considering genetic drift give us a good prediction of the expected proportion of correct alleles when using exponentially scaled representations. Population sizing models that neglect the effect of genetic drift are not able to accurately predict the expected proportion of correct alleles for a given population size.

3.2.5 Conclusions

We have illustrated the effect of non-uniformly scaled representations on the performance of genetic algorithms. When using small populations and easy problems, GAs are affected by genetic drift. To be able to model the effects of genetic drift more accurately, we used the population sizing model from Harik et al. (1999), the domino convergence model from Rudnick (1992), and the time complexity model from Thierens (1995) and Thierens et al. (1998) and developed two population sizing models for exponentially scaled representations considering genetic drift. The approximated genetic drift model uses an approximation (Kimura 1964) for the probability that an allele is completely converged due to genetic drift after t generations, and gives us a lower bound for the solution quality. The stair-case drift model assumes that genetic drift occurs as long as the convergence time is larger than the expected drift time, and that the lower salient genes are fixed at the correct solution due to genetic drift with probability x_0/N.

The theoretical results reveal that genetic drift has a large impact on the probability of error and convergence time when using exponentially scaled representations. Because the alleles are solved strictly in serial, exponentially scaled representations change the dynamics of genetic search. As a result the solution quality is reduced by genetic drift, and the convergence time is increased by domino convergence. The empirical investigations show that despite the assumption that the size of the convergence window $\lambda_c = 1$, the proposed models considering genetic drift give us, in contrast to the no-drift model, accurate predictions for the solution quality. Except for a very large number m of competing BBs, or a very low number l_s of exponentially scaled genotypic alleles, the no-drift population sizing model is not able to predict the expected solution quality.

When using exponentially scaled representations, researchers should be aware of the effects of genetic drift as some of the alleles are fixed randomly and of the effects of domino convergence which increases the time to convergence.

3.3 Locality

During the last decades, starting with work by Bagley (1967), Rosenberg (1967) and Cavicchio (1970), researchers recognized that the concept of building blocks is helpful for understanding the principles of selectorecombinative GAs, and is a key factor in determining successful use. A well designed GA should be able to preserve high quality building blocks, and increase their number over the generations (Goldberg 1989c).

When using the notion of building blocks in the context of representations, we must be aware that building blocks not only exist in the genotype, but also in the phenotype. A representation transforms the structure and complexity of the building blocks from the phenotype to the genotype, and therefore the structure and complexity of the building blocks can be different in the genotype and phenotype. This section provides the third, and final element, of a theory of representations for GEAs and investigates how the locality of a representation modifies the complexity of building blocks and changes the difficulty of an optimization problem. The locality of a representation describes how well genotypic neighbors correspond to phenotypic neighbors. The locality of a representation is high if genotypic neighbors correspond to phenotypic neighbors.

Previous work has indicated that high locality of a representation is necessary for efficient evolutionary search (Rothlauf and Goldberg 1999; Gottlieb and Raidl 1999; Gottlieb and Raidl 2000; Gottlieb et al. 2001; Rothlauf and Goldberg 2000). However, it remains unclear as to how the locality of a representation influences problem difficulty, and if high-locality representations always aid evolutionary search. The results show that high-locality representations do not modify the complexity of the problems they are used for. In contrast, when using low-locality representations the genotypic BBs can be different from the phenotypic BBs, and the complexity of the problem can be changed. Therefore, only high-locality representations can guarantee to reliably and predictably solve problems of bounded complexity.

We focus our investigation on how the locality of a representation influences problem difficulty and do not develop models for problem difficulty but use selected models presented in Sect. 2.3. In the context of evolution strategies (Bäck and Schwefel 1995), previous work developed the concept of causality (Igel 1998; Sendhoff et al. 1997b; Sendhoff et al. 1997a) as a measurement of problem difficulty. Basically, both concepts, causality and locality, address similar aspects as they describe how well the distances between individuals are preserved when mapping the phenotypes on the genotypes. However, in this study we do not classify the difficulty of problems but only describe how the difficulty of a problem is changed by the encoding.

In the following subsection, the influence of representations on problem difficulty is discussed. We illustrate why it is helpful to use encodings that do not modify problem difficulty for solving problems of bounded difficulty. Section 3.3.2 introduces the concept of locality and shows how locality depends on

the metrics that are defined for the genotypes and phenotypes. In Sects. 3.3.3 and 3.3.4, we review the concept of problem difficulty introduced by Jones and Forrest (1995), and show how the locality of a representation influences problem difficulty. Sect. 3.3.5 introduces the distance distortion of a representation and illustrates that high locality is necessary for an encoding to have low distance distortion. In Sect. 3.3.6, we show for the fully easy one-max problem that low-locality representations make the problem more difficult. The results of the theoretical investigation are verified and illustrated in Sect. 3.3.7 by an empirical investigation for the one-max and deceptive trap problem. The results show that for high-locality representations the one-max problem remains fully easy, and the fully deceptive trap remains fully difficult. In contrast, when using low-locality representations the fully easy one-max problem becomes more difficult to solve for GEAs, whereas the deceptive trap becomes easier. The section ends with concluding remarks.

3.3.1 Influence of Representations on Problem Difficulty

We discuss the influence of representations on problem difficulty and why representations should preserve the complexity of a problem when assigning the genotypes to the phenotypes.

It was already recognized by Liepins and Vose (1990) that by using different representations the complexity of a problem can be completely changed. Changing the complexity of a problem means that the difficulty of the problem and therefore the structure of the BBs is changed (see Sect. 2.3). However, representations that modify BB-complexity are problematic as they do not allow us to reliably solve problems of bounded difficulty. If the complexity of BBs is changed by the encoding, some problems become easier to solve, whereas others become more difficult. To predict which problems become easier and which do not is only possible if detailed knowledge about the optimization problem exists (compare also Sects. 3.3.4 and 4.4.3). To ensure that problems of bounded complexity, that means easy problems, can be reliably solved by GEAs, representations that preserve the complexity of the building blocks should be used.

Liepins and Vose (1990) showed that for a fully deceptive problem $f(\boldsymbol{x}) = f_p(f_g(\boldsymbol{x}))$ there is a transformation T such that the function $g(\boldsymbol{x}) = f[T(\boldsymbol{x})]$ becomes fully easy. This means that every fully deceptive problem could be transformed into a fully easy problem by introducing a linear transformation T. In general, the difficulty of a problem could easily be modified by using different linear transformations T. Liepins and Vose concluded that their results underscore the importance of selecting good representations, and that good representations are problem-specific. Therefore, in order to find the best representation for a problem, it is necessary to know how to optimize the problem and what the optimal solution for the problem is.

To find the best representation for a specific problem, Liepins and Vose proposed using adaptive representations that "simultanously search for rep-

resentations at a metalevel" (Liepins and Vose 1990, p. 110). These representations should autonomously adapt to the problem and encode it in a proper way (compare Rothlauf et al. (2000)). Initial steps in this direction were made by Goldberg et al. (1989) with the development of the messy GA. These kinds of GAs use an adaptive encoding that adapts the structure of the representation to the properties of the problem. This approach, however, burdens the GA not only with the search for promising solutions, but also the search for a good representation.

Therefore, we go back one step and ask the question, what kind of encoding should be used if there is no a priori knowledge about the problem, and no adaptive encoding should be used. Users who simply want to solve their problems by using a GEA are confronted with this kind of problem. They have no knowledge about the problem, and they do not want to do experiments to find out what structure the problem has, what the promising areas in the search space are, and what kind of representations make the problem most easy. They just want to be sure that the GEA can solve their problem as long as the complexity of their problem is bounded and the used GEA is able to solve it. One solution for their problems are representations that preserve BB-complexity. Using these types of representations means that the problem has the same difficulty in the genotypic as in the phenotypic space. Then users can be sure that the representation does not increase the complexity of the problem, and that the GEA reliably solves their problem.

Wanting representations to preserve BB-complexity raises the question of, why are we especially interested in encodings that preserve complexity? Is it not more desirable to construct encodings that reduce problem complexity, as Liepins and Vose propose? The answer is yes and no. Of course, we are interested in encodings that reduce problem difficulty. However, in general it is not possible to construct a representation that reduces the complexity for all possible problem instances. One result of the no-free-lunch theorem (Wolpert and Macready 1995) is that if some problem instances become easier by the use of a representation, there are other types of problem instances that necessarily become more difficult.

Therefore, we want to at least ensure that the representation does not make problems of bounded difficulty – these are the class of problems we are interested in – more difficult to solve. However, as shown in Sects. 3.3.6 and 3.3.7, encodings that do not preserve problem complexity always make fully easy problems more difficult. Therefore, a phenotypically easy problem could even become so difficult by using a "bad" encoding that it can not be solved efficiently any more. Of course, representations that do not preserve problem complexity make fully difficult problems more easy, but as we are interested in solving only problems of bounded difficulty, and not all types of problems, this is not important to us. For difficult problems, like the needle in the haystack problem, or the fully deceptive trap problem, the complexity of the problem is not bounded, and therefore, we are in general not interested in solving these kinds of problems with GEAs.

The solution for all our problems would be an "optimal" representation that preserves problem complexity for problems of bounded difficulty *and* reduces the complexity for all other problems. But reaching this aim is far beyond the scope of this work. Furthermore, we believe that the no no-free-lunch theorem does not allow us to get such a free lunch for every problem.

Finally, we want to come back to the results of Liepins and Vose (1990) and illustrate the problems with representations that change the difficulty of problems. The transformation T, which can be interpreted as a genotype-phenotype mapping, can modify the complexity of a problem in such a way that a fully difficult deceptive trap problem becomes a fully easy one-max problem. But, using the same transformation T for a fully easy one-max problem can result in a fully deceptive trap (compare Fig. 4.1(a)). Therefore, by using this representation, we are able to solve a deceptive trap, but not the one-max problem any more. If we want to solve both types of problems we must know a priori what the problem is and adjust the representation according to the problem. However, if we do not know the problem a priori, and if we want to make sure that we can solve at least problems of bounded difficulty reliably, we must use representations that do not modify problem difficulty.

In the following subsection, we define the locality of a representation formally and show in Sect. 3.3.4 that low locality is necessary for a representation to not modify problem difficulty.

3.3.2 Metrics, Locality, and Mutation Operators

In Sects. 2.1.1 and 2.1.2, we introduced the concept of a representation which assigns genotypes $x^g \in \Phi_g$ to corresponding phenotypes $x^p \in \Phi_p$. In the following paragraphs, we introduce the concept of locality and describe how the locality of a representation is based on the metric used for Φ_g and Φ_p.

When using search algorithms, a metric has to be defined on the search space Φ. Based on the metric, the distance d_{x_a,x_b} between two individuals $x_a \in \Phi$ and $x_b \in \Phi$ describes how similar the two individuals are. The larger the distance, the more different two individuals are. In general, different metrics can be defined for the same search space. Different metrics result in different distances and different measurements of the similarity of solutions.

Two individuals are neighbors if the distance between two individuals is minimal. For example, when using the Hamming metric (Hamming 1980) for binary strings the minimal distance between two different individuals is $d_{min} = 1$. Therefore, two individuals x_a and x_b are neighbors if the distance $d_{x_a,x_b} = 1$.

If we use a representation f_g there are two different search spaces, Φ_g and Φ_p. Therefore, different metrics can be used for the phenotypic and the genotypic search space. In general, the metric used on the phenotypic search space Φ_p is determined by the specific problem that should be solved and describes which problem solutions are similar to each other. In contrast, the metric defined on Φ_g is not given a priori but depends on the used genotypes.

As different genotypes can be used to represent the same phenotypes, different metrics can be defined on Φ_g. Therefore, in general, different metrics are used for Φ_p and Φ_g which imply a different neighborhood structure in Φ_g and Φ_p. For example, when encoding integers using binary strings the phenotype $x^p = 5$ has two neighbors, $y^p = 6$ and $z^p = 4$. When using the Hamming metric, the corresponding binary string $x^g = 101$ has three different neighbors, $a^g = 001$, $b^g = 111$, and $z^g = 100$ (Caruana and Schaffer 1988).

The locality of a representation describes how well neighboring genotypes correspond to neighboring phenotypes. The locality of a representation is high if all neighboring genotypes correspond to neighboring phenotypes. In contrast, the locality of a representation is low if some neighboring genotypes do not correspond to neighboring phenotypes. Therefore, the locality d_m of a representation can be defined as

$$d_m = \sum_{d^g_{x,y}=d^g_{min}} |d^p_{x,y} - d^p_{min}|, \tag{3.23}$$

where $d^p_{x,y}$ is the phenotypic distance between the phenotypes x^p and y^p, $d^g_{x,y}$ is the genotypic distance between the corresponding genotypes, and d^p_{min}, resp. d^g_{min} is the minimum distance between two (neighboring) phenotypes, resp. genotypes. Without loss of generality we want to assume that $d^g_{min} = d^p_{min}$. For $d_m = 0$ all genotypic neighbors correspond to phenotypic neighbors and the encoding has perfect (high) locality.

We want to emphasize that the locality of a representation does not only depend on the representation f_g, but also on the metric that is defined on Φ_g and the metric that is defined on Φ_p. f_g only determines which phenotypes are represented by which genotypes and says nothing about the similarity between solutions. Before we are able to describe the locality of a representation a metric must be defined on Φ_g and Φ_p.

In the remaining paragraphs, we briefly discuss how the mutation operator used for genetic search determines the metric and the distances that are used for the genotypic space Φ_g. Based on the metric defined on the genotypic search space Φ_g, search operators like mutation or crossover can be defined. In EAs, and most of the individual-based search heuristics, like simulated annealing, tabu search, and others, the search operator mutation is designed to create new solutions (offspring) with similar properties as its/their parent(s) (Doran and Michie 1966). In most local search methods, mutation creates offspring that have a small or sometimes even minimal distance to their parents (for example the bit-flipping operator for binary representations). Therefore, mutation operators and metrics can not be developed independently of each other but determine each other. A metric defines the mutation operator and the used mutation operator determines the metric. As the search operators are applied to the genotypes, the metric that is used on Φ_g is relevant for the definition of mutation operators.

The basic idea behind using mutation-based search approaches is that the structure of the fitness landscape should guide the search heuristic to the high-quality solutions (Manderick et al. 1991), and that the optimal solution can be found by performing small iterated changes. It is assumed that the high-quality solutions are not isolated in the search space but grouped together (Christensen and Oppacher 2001; Whitley 2002). Therefore, better solutions can easily be found by searching around already found good solutions. The search steps must be small because too large search steps would result in a randomization of the search, and guided search around good solutions would become impossible. In contrast, when using search operators that perform large steps in the search space it would not be possible to find better solutions by searching around already found good solutions but the search algorithm would jump randomly around the search space (compare also Sect. 3.1.2).

3.3.3 Phenotype-Fitness Mappings and Problem Difficulty

As described in Sect. 2.1.2 the difficulty of a problem depends on the phenotype-fitness mapping f_p. Furthermore (compare Sect. 2.3), the difficulty of a problem depends on the used search method and the metric that is defined on the phenotypic search space Φ_p. The metric defined on Φ_p determines which individuals are similar to each other and depends on the used main search operator. Both determinants of problem difficulty, the phenotype-fitness mapping f_p and the metric defined on Φ_p, are given a priori by the character of the optimization problem that should be solved and by the used search method.

In Sect. 3.3.2, we described that the mutation operator and the used metric determine each other. Different mutation operators imply different metrics. As problem difficulty depends not only on f_p but also on the metric defined on Φ_p the difficulty of a problem is not absolute but depends on the used metric respectively search operator. The use of different metrics and search operators result in a different problem difficulty. Consequently, the difficulty of a problem can only be defined with respect to a search operator. It makes no sense to say a problem is either easy or difficult if the used search operator is not taken into account.

In Sect. 2.3.2, we gave a short review of some measurements for problem difficulty in the context of GEAs. In the following subsections, we want to use the classification of problem difficulty from Jones and Forrest (1995), which is based on the correlation analysis, for describing how the locality of a representation influences GEA performance. The difficulty of an optimization problem is determined by how the fitness values are assigned to the phenotypes and what metric is defined on the phenotypes. Combining both aspects we can measure problem difficulty by the fitness-distance correlation coefficient $\rho_{FDC} \in [-1, 1]$ of a problem (Jones 1995; Jones and Forrest 1995). ρ_{FDC} measures the correlation between the fitnesses of search points and their dis-

tances to the global optimum. We want to distinguish between three different classes of problem difficulty:

1. The fitness difference to the optimal solution is positively correlated with the distance to the optimal solution. With lower distance the fitness difference to the optimal solution decreases. As the structure of the search space guides local search methods to the optimal solution such problems are easy for mutation-based search.
2. There is no correlation between the fitness difference and the distance to the optimal solution. The fitness values of neighboring individuals are uncorrelated and the structure of the search space provides no information about which solutions should be sampled next by the search method.
3. The fitness difference is negatively correlated to the distance to the optimal solution. Therefore, the structure of the search space misleads a local search method to sub-optimal solutions.

The three different classes of problem difficulty are illustrated in Fig. 3.13. We show how the fitness difference $|f_{opt} - f|$ depends on the distance d to the optimal solution. In the following paragraphs, we want to discuss these three classes in some more detail.

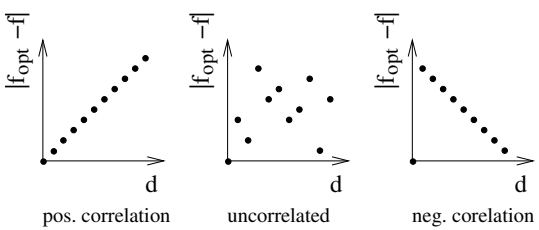


Figure 3.13. Different classes of problem difficulty

Problems are easy for mutation-based search if there is a positive correlation between an individuals' distance to the optimal solution and the difference between its fitness and the fitness of the optimal solution. Many test problems that are commonly used for EAs like the sphere and corridor models for evolution strategies or the one-max problem for genetic algorithms show this behavior. Such problems are easy for local and crossover-based search methods as the search is guided to the optimal solution by the structure of the fitness landscape.

Problems become much more difficult if there is no correlation between the fitness difference and the distance to the optimal solution. Then, the fitness landscape does not guide a mutation-based search method to the optimal solution. No search heuristics can use information about a problem which was collected in prior search steps to determine the next search step. Therefore, all reasonable search algorithms show the same performance as no useful information (information that indicates where the optimal solution can be found)

is available in the problem. Because all search strategies are equivalent, also random search is an an appropriate search method for such problems. Random search uses no information and performs well on these types of problems.

Problem difficulty is maximal for mutation-based search methods if the fitness landscape leads the search method away from the optimal solution. Then, the distance to the optimal solution is negatively correlated to the fitness difference between an individual and the optimal solution. Because mutation-based search finds the optimal solution by performing iterated small steps in the direction of better solutions, all mutation-based search approaches must fail as they are mislead. All other search methods that use information about the fitness landscape also fail. The most effective search methods for such problems are those that do not use information about the structure of the search space but search randomly like random search. The most prominent example for such types of problems are deceptive traps. Such problems are commonly used to perform a worst-case analysis for EAs.

Although we use this problem classification for investigating the influence of locality on problem difficulty, we want to emphasize that in general this problem classification is not relevant for most of the real-world problem instances. Only problems of class one can be solved efficiently using EAs or local search as this problem class guides the local search methods (like mutation-based EAs) to the good solutions. In contrast, for problems of class two, mutation-based search methods perform the same as random search, and for problems of class three random search performs even better. This situation is not in contrast to the observed good performance of EAs on many real-world problem instances. EAs show a good performance as most of the real-world problems are easy problems and belong to class one (compare also Sect. 3.3.1). In general, for real-world problems the fitness values of neighboring solutions are correlated, and high-quality and low-quality solutions are grouped together. Therefore, fitness landscapes that are uncorrelated, or even deceptive, are uncommon in real world.

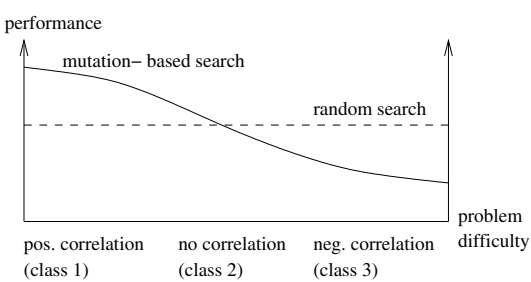


Figure 3.14. Performance of mutation-based EA search versus random search

This situation is illustrated in Fig. 3.14. We know from the no-free-lunch theorem that all search methods show on average the same performance over all possible problem instances (Wolpert and Macready 1995; Whitley 2000a).

Furthermore, we know that the performance of random search remains constant over all problem instances and that mutation-based evolutionary search performs well on problems of class one. Consequently, it must show low performance on other problem instances (class 3). As the performance of mutation-based search is "biased" towards problems of class one, many real-world instances can efficiently be solved using mutation-based EAs.

3.3.4 Influence of Locality on Problem Difficulty

In Sect. 3.3.1, we described how representations can change the character and difficulty of optimization problems. In the following paragraphs, we discuss high versus low-locality representations and examine how the locality of a representation influences problem difficulty.

Low versus High-Locality Representations

We have seen in Sect. 3.3.2 that the metric defined on Φ_p can be different from the metric defined on Φ_g. As the locality of a representation measures how well the phenotypic metric corresponds to the genotypic metric, it is possible to distinguish between high and low-locality representations. Representations have high locality if neighboring genotypes correspond to neighboring phenotypes. In contrast, representations have low locality if neighboring genotypes do not correspond to neighboring phenotypes. Figure 3.15 illustrates the difference between high and low-locality representations. In this example, we assume that there are 12 different phenotypes (a-l) and that there is a metric defined on Φ_p (in this example the Euclidean distance). Each phenotype (lower case symbol) corresponds to one genotype (upper case symbol). The representation f_g has perfect (high) locality if neighboring phenotypes correspond to neighboring genotypes. Then a mutation step has the same effect in the phenotypic and genotypic search space.

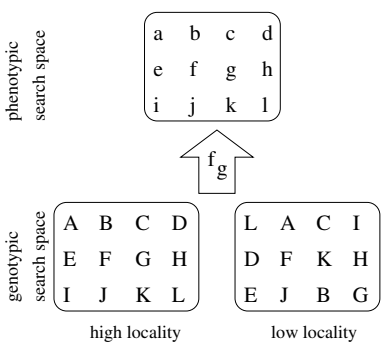


Figure 3.15. High versus low-locality representations

As we assume in our considerations that f_g is a one-to-one mapping every phenotype is represented by exactly one genotype and there are $|\Phi_g|! = |\Phi_p|!$

different representations. $|\Phi_g|$ is the size of the genotypic search space. Each of these many different representations assigns the genotypes to the phenotypes in a different way. A common example are different representations for representing integer phenotypes using binary strings. Both, binary and Gray encoding, represent integers using binary strings of the same length but they differ in which phenotype is represented by which genotype.

Investigating the relationship between different representations (how the genotypes are assigned to the phenotypes) and the used mutation operator (which is based on the genotypic metric) reveals that a different assignment of genotypes to phenotypes can be equivalent to the use of a different metric for Φ_g. This effect is known as the isomorphism of fitness landscapes (Reeves 1999). For example, it can be shown that the use of a simple bit-flipping operator (which induces the Hamming metric) for Gray encoded problems is equivalent to the use of the complementary crossover operator (which induces a different "non-Hamming" metric) for binary encoded problems (Reeves 2000). Both metric-representation combinations result in the same fitness landscape and therefore in the same performance of mutation-based search.

Influence on Problem Difficulty

In the following paragraphs, we examine how the locality of a representations influences the performance of GEAs. A representation transforms the phenotypic problem f_p with a given phenotypic problem difficulty into a genotypic problem $f = f_p \circ f_g$ with a resulting genotypic problem difficulty that can be different from the phenotypic problem difficulty. We use the problem classification described in Sect. 3.3.3.

We have seen that the phenotypic difficulty of an optimization problem depends on the metric that is defined on the phenotypes and the function f_p which assigns a fitness value to every phenotype. Based on the phenotypic metric a local or crossover-based search operator can be defined (for the phenotypes). By the use of a representation, which assigns a genotype to every phenotype, a new genotypic metric is introduced which can differ from the phenotypic metric. Therefore, the character of the search operator can also be different for genotypes and phenotypes. If the locality of a representation is high, then the search operator has the same effect on the phenotypes as on the genotypes. As a result, genotypic and phenotypic problem difficulty is the same and the difficulty of a problem remains unchanged by the use of an additional representation f_g. Easy phenotypic problems remain genotypically easy (compare the results presented in Fig. 3.21) and difficult phenotypic problems remain genotypically difficult (compare Fig. 3.22). Figure 3.16 (left) illustrates the effect of mutation for high-locality representations. The search operator (mutation) has the same effect on the phenotypes as on the genotypes.

The situation is different when focusing on low-locality representations. Here, the influence of the representation on the difficulty of a problem depends on the considered optimization problem. If the considered problem f_p

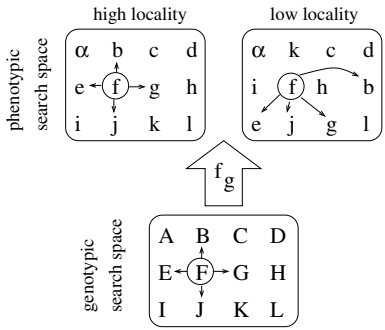


Figure 3.16. The effect of mutation for high versus low-locality representations

is easy (class 1) and the structure of the search space guides the search method to the optimal solution, a low-locality representation f_g randomizes the problem and makes the overall problem f more difficult. When using low-locality representations a small change in a genotype does not correspond to a small change in the phenotype, but larger changes in the phenotype are possible (compare Fig. 3.16 (right)). Therefore, when using low-locality representations, phenotypic easy problems of class one become on average genotypic problems of class two. Low-locality representations lead to a more uncorrelated fitness landscape and heuristics can no longer extract information about the structure of the problem. Guided search becomes more difficult as many genotypic search steps do not result in a similar individual but in a random one.

If the problem f_p is of class two, on average a low-locality representation does not change the problem class. Although the mutation-based search becomes more random search, the performance stays constant as random search and mutation-based search show the same performance for problems of class two. Of course, representations exist that can make a problem easier and result in an overall genotypic problem f of class one; however, there are only few of these and most of the low-locality representations simply modify the problem and do not create a fitness landscape of class one which leads the search method to the good solutions. On the other hand, there are also representations f_g that construct a problem f which misleads mutation-based search and transforms a problem of class two into class three. But as for low-locality representations that transform a problem from class two into class one, there are only few such representations.

Finally, we have to consider problems of class three. On average, the use of low-locality representations transforms such problems into problems of class two as the problems become more randomized. Then, mutation-based search is less misled by the fitness landscape and the problem difficulty for mutation-based search is reduced. On average, low-locality representations "destroy" the deceptiveness of class three problems and turn them into problems of class two.

Summarizing the results, we recognize that low-locality representations have the same effect as when using random search. Therefore, on average problems of class one become more difficult, and problems of class three more easy to solve. As most real-world problems belong to class one, the use of low-locality representations makes these problems more difficult. Therefore, we strongly encourage researchers to use high-locality representations for problems of practical relevance. Of course, low-locality representations make deceptive problems easier; however, these are problems which we do not expect to meet in reality and are only of theoretical interest.

3.3.5 Distance Distortion and Crossover Operators

We extend the notion of locality and introduce the *distance distortion* of an encoding. The concept of distance distortion is related to the concept of heritability which describes that a crossover operator should create new offspring that have similar properties to their parents. Appropriate measurements for heritability describe how well offspring take over advantageous features of their parents (Gottlieb and Raidl 1999).

When using recombination-based search, the locality concerning small changes d_m can be extended towards locality concerning small and large changes. The distance distortion d_c describes how well the phenotypic distance structure is preserved when mapping Φ_p on Φ_g:

$$d_c = \frac{2}{n_p(n_p-1)} \sum_{i=1}^{n_p} \sum_{j=i+1}^{n_p} |d^p_{x,y} - d^g_{x,y}|,$$

where n_p is the number of different individuals, $n_p = |\Phi_g| = |\Phi_p|$, and $d^g_{min} = d^p_{min}$. For $d_c = 0$ all phenotypic distances are preserved by the representation. We see that for $\Phi_g = \Phi_p$ high locality ($d_m = 0$) results in low distance distortion ($d_c = 0$). If, for example, our genotypic and phenotypic search space is binary, and the locality of the genotype-phenotype mapping is perfect, then all distances between the individuals are preserved. However, if we assume that $\Phi_g \neq \Phi_p$, then high locality is a necessary, but not sufficient condition for the genotype-phenotype mapping to have low distance distortion.

Figure 3.17 illustrates the difference between representations with high versus low distance distortion. The distance distortion d_c of a representation is low if the genotypic distances correspond to the phenotypic distances. If the distances between the genotypes and the corresponding phenotypes are different, then the distance distortion d_c of the representation is high.

It is of interest that the locality d_m and distance distortion d_c do not require the definition of genetic operators a priori. It is sufficient to define both based on the distance metrics used for Φ_g and Φ_p. The application of mutation to an individual should result in an offspring that is similar to its parent. Therefore, in many implementations, mutation creates offspring who have the lowest possible distance to the parent (for example the bit-flipping

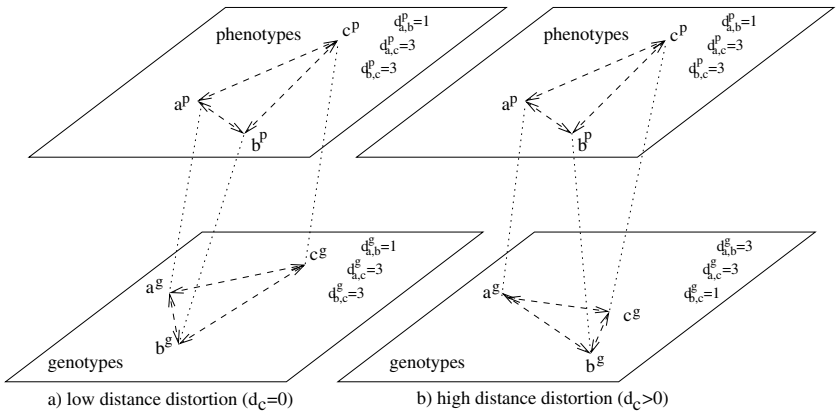


Figure 3.17. The figures illustrate the difference between representations with low versus high distance distortion. If the distance distortion $d_c = 0$ then the distances between the corresponding genotypes and phenotypes are the same. If $d_c > 0$ genotypic distances between individuals do not correspond to phenotypic distances between individuals.

operator for binary representations). High locality of a representation is a necessary condition for successful use of mutation-based search algorithms. Otherwise, low-locality encodings do not allow a guided search and GEAs using low-locality representations behave like random search.

The situation is similar when using crossover operators. The application of crossover operators should result in offspring where the distances between the offspring and its two parents are smaller than the distance between both parents. Common standard crossover operators, like n-point or uniform crossover show this behavior. The distances between genotypic offspring and parents are always lower, or equal to, the distances between both parents. However, if a representation has high distance distortion, the genotypic distances do not correspond to the phenotypic distances. Then, the phenotypic distances between offspring and parents are not necessarily smaller than the phenotypic distances between both parents. The application of crossover to genotypes does not result in offspring phenotypes that mainly consist of substructures of their parents' phenotypes. Therefore, the offspring is not similar to its parents and the use of crossover results in random search. We see that low distance distortion of a representation is a necessary condition for good performance of crossover-based GAs.

Examining the interdependencies between locality and distance distortion shows that high locality is a necessary condition for an encoding to have low distance distortion. When using standard crossover operators such as uniform or n-point crossover, the offspring could not inherit the properties of the parents if similar genotypes result in completely different phenotypes. If the encoding has low locality, the crossover operators would create offspring

genotypes which are similar to the genotypes of the parents, but the resulting phenotypes would not be similar to the phenotypes of the parents. Thus, low locality of a representation would also result in high distance distortion.

In the following subsection, we show for the fully easy one-max problem that the problem only stays fully easy if a high-locality representation is used and $d_m = d_c = 0$. All other types of representations increase the difficulty of the problem.

3.3.6 Modifying BB-Complexity for the One-Max Problem

In this subsection, we investigate the influence of locality on GA performance. We show for the fully easy one-max problem (3.24) that the use of a high-locality representation does preserve problem difficulty whereas other types of representations reduce GA performance.

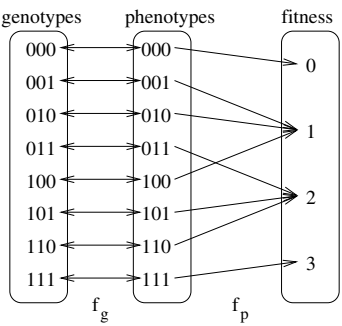


Figure 3.18. A representation for the bit-counting function with perfect locality and no distance distortion

For the one-max, or bit-counting problem, the function $f_p : \{0,1\}^l \to \mathbb{R}$ assigns to every individual $x^p \in \Phi_p$ the fitness value $\sum_{i=0}^{l-1} x_i^p$. As there are only l fitness values that are assigned to 2^l phenotypes the fitness function f_p is affected by redundancy (see Sect. 3.1.6). The genotype-phenotype mapping f_g is a non-redundant one-to-one mapping, and the genotypic space Φ_g, and the phenotypic space Φ_p, have the same size $|\Phi_g| = |\Phi_p| = 2^l$ and the same properties $\Phi_g = \Phi_p$. To simplify the investigation we want to assume, without loss of generality, that the phenotype with only ones is always represented by the genotype with only ones, and therefore is always the global optimum. In Fig. 3.18, a 3-bit one-max problem is illustrated. The encoding used, which can be described by the genotype-phenotype mapping f_g, has high locality ($d_m = 0$) and preserves the distances between the individuals when mapping the phenotypes to the genotypes ($d_c = 0$) as the genotype-phenotype mapping is the identity mapping $x^p = f_g(x^g) = x^g$. As a result, the phenotypic and genotypic problem complexity is the same.

We investigate how problem difficulty changes if we use a representation where $d_c \neq 0$. For measuring problem difficulty we use the fitness of the

schemata (compare Sect. 2.3.2). The fitness of a schema h of size λ is defined as $f(h) = f(u, \lambda, l)$. It has length l, u ones, $\lambda - u$ zeros in the fixed positions, and $l - \lambda$ don't care positions. For the one-max problem the schema fitness in terms of the function values $f(u) = u$ can be calculated as follows:

$$f(u, \lambda, l) = \frac{1}{2^{l-\lambda}} \sum_{i=0}^{l-\lambda} \binom{l-\lambda}{i} (i + u).$$

For all schemata of size λ the difference between the fitness $f(\lambda, \lambda, l)$ of the best schemata with λ ones, and the fitness $f(\lambda - 1, \lambda, l)$ of its strongest competitor with $\lambda - 1$ ones is

$$d = \frac{1}{2^{l-\lambda}} \sum_{i=0}^{l-\lambda} \binom{l-\lambda}{i} > 0.$$

Thus, all schemata h that contain the global optimum x^{opt} (a string of only ones) are superior to their competitors, and the one-max problem is phenotypically fully easy.

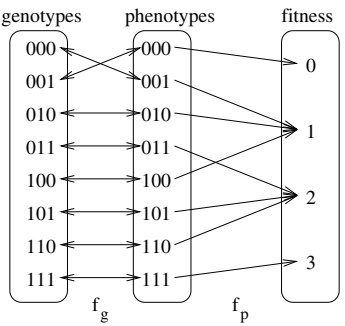


Figure 3.19. A low-locality representation for the one-max problem which does not preserve the distances between the individuals ($d_c \neq 0$), and therefore modifies BB-complexity. The distance distortion $d_c = \frac{2}{7*8} * 12 = 3/7 \neq 0$.

We investigate how the problem complexity changes if the distances between the individuals are changed by using a low-locality representation. For example, the genotype-phenotype mapping of two genotypes x^g and y^g which correspond to the phenotypes x^p and y^p, is changed such that afterwords $x^g = 000$ represents $y^p = 001$, and $y^g = 001$ represents $x^p = 000$ (see Fig. 3.19). Beginning with the high-locality encoding illustrated in Fig. 3.18, there are three different possibilities when changing the mapping of two individuals:

- Both individuals x^g and y^g have the same number of ones ($u_{x^g} = u_{y^g}$)
- Both individuals have a different number of ones, and the number of different positions d_{x^g,y^g} in the two individuals x^g and y^g is the same as the number of different ones ($d_{x^g,y^g} = u_{x^g} - u_{y^g}$)
- Both individuals have a different number of ones, and the number of different positions is higher than the number of different ones ($d_{x^g,y^g} > u_{x^g} - u_{y^g}$)

In the following paragraphs, we investigate how the difficulty (measured using the notion of schemata) of the problem is changed for these three situations.

If f_g is modified for two genotypes x^g and y^g that have the same number of ones in the string, then the corresponding fitness values remain unchanged ($f(x^g) = f(y^g)$). Therefore, the fitness of the schemata, and the difficulty of the problem both remain constant. For example, we can change the mapping of the genotypes 1001 and 0101 for a 4 bit one-max problem. Both individuals have two ones in the string and their fitness is two. The difficulty of the problem remains unchanged.

f_g could be modified for two individuals x^g and y^g that have a different number of ones, and therefore different fitness values. We assume that the number of different positions in the two individuals is the same as the number of different ones ($d_{x^g,y^g} = u_{x^g} - u_{y^g}$). Before the change, the individual x^g has l ones and therefore fitness l; y^g has h ones and fitness h. We want to assume $h > l$. After the change, x^g has fitness h although it has only l ones, whereas y_g has only a fitness of l but h ones. Before the modification, all schemata that lead to the global solution are superior to their competitors. Subsequently, after the modification of the mapping the fitness of all schemata h that contain y^g but not x^g, is reduced by $(h-l)/(2^{l-\lambda})$, whereas the fitness of all misleading schemata containing only x^g is increased by this amount. Schemata that contain x^g as well as y^g are not changed. As a result, the average fitness of high quality schemata is reduced, whereas the fitness of misleading schemata is increased. Let us illustrate this with a small 3-bit example. f_g from Fig. 3.18 should be modified for the genotypes 001 and 101. Therefore, $x^g = 001$ corresponds to $x^p = 101$ and $y_g = 101$ corresponds to $y_p = 001$ Then, individual $x^g = 001$ has fitness 2, and individual $y_g = 101$ has fitness 1. The fitness of the schema 1** is reduced, whereas the fitness of schema 0** increases. For size two schemata, the fitness of 10* and 1*1 decreases, whereas the fitness of 00* and 0*1 increases. As a result, the problem becomes more difficult to solve for a GA.

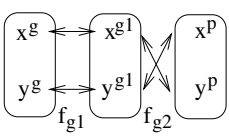


Figure 3.20. A decomposition of f_g

Finally, we could decompose f_g into two mappings f_{g1} and f_{g2} if the number of different positions in the two genotypes x^g and y^g is higher than the number of different ones $d_{x^g,y^g} > u_{x^g} - u_{y^g}$ (see Fig. 3.20). f_{g1} maps x^g to x^{g1}, and y^g to y^{g1}. x^{g1} (resp. y^{g1}) should have the same number of ones as x^g (resp. y^g) ($u_{x^{g1}} = u_{x^g}$, $u_{y^{g1}} = u_{y^g}$), but some positions are different in the two individuals x^{g1} and y^{g1} ($d_{x^{g1},y^{g1}} = u_{x^p} - u_{y^p}$). Therefore, as the number of ones stays constant, f_{g1} does not change the fitness of the schemata (compare item 1). For x^{g1} and x^p (resp. y^{g1} and y^p), the number of different ones is the

same as the number of different positions. Thus, f_{g2} has the same properties as discussed in the previous item and increases the fitness of misleading schemata, as well as reduces the fitness of the high-quality schemata.

We see that most modifications of a high-locality genotype-phenotype mapping f_g make the one-max problem more difficult to solve. Only when the mapping between genotypes and phenotypes is changed that have the same number of ones in the string, is the structure of the BBs preserved, and we get the same performance as for the high-locality representation from Fig. 3.18. The above proof can be applied in the same way to a fully deceptive trap problem. Then, most of the low-locality encodings reduce the fitness of the misleading schemata, and increase the fitness of the high-quality schemata, which makes the problem easier. In the following subsection, we present an empirical verification of the results.

3.3.7 Empirical Results

In this subsection, we present an empirical investigation into how the problem complexity is changed for the one-max problem and the deceptive trap problem if low-locality representations are used. We experimentally show that for high-locality representations the fully easy one-max problem remains fully easy. In contrast, most of the low-locality representations make the one-max problem more difficult to solve for GAs. The situation is vice versa for the fully difficult deceptive trap where low-locality representations always makes the problem easier to solve.

For a non-redundant genotype-phenotype mapping f_g that is defined on binary genotypes of length l, there are $2^l!$ different possibilities to assign the 2^l genotypes to the 2^l phenotypes (Assigning the genotypes to the phenotypes can be interpreted as a permutation of 2^l numbers). Any of these possibilities represents a specific mapping like for example the binary encoding or the Gray encoding.

Table 3.6. 24 possibilities to assign four genotypes $\{a^g,\ b^g,\ c^g,\ d^g\}$ to four phenotypes $\{a^p,\ b^p,\ c^p,\ d^p\}$

x^p	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
a^p	a^g	a^g	a^g	a^g	a^g	a^g	b^g	b^g	b^g	b^g	b^g	b^g	c^g	c^g	c^g	c^g	c^g	c^g	d^g	d^g	d^g	d^g	d^g	d^g
b^p	b^g	b^g	c^g	c^g	d^g	d^g	a^g	a^g	c^g	c^g	d^g	d^g	a^g	a^g	b^g	b^g	d^g	d^g	a^g	a^g	b^g	b^g	c^g	c^g
c^p	c^g	d^g	b^g	d^g	b^g	c^g	c^g	d^g	a^g	d^g	a^g	c^g	b^g	d^g	a^g	d^g	a^g	b^g	b^g	c^g	a^g	c^g	a^g	b^g
d^p	d^g	c^g	d^g	b^g	c^g	b^g	d^g	c^g	d^g	a^g	c^g	a^g	d^g	b^g	d^g	a^g	b^g	a^g	c^g	b^g	c^g	a^g	b^g	a^g

In Table 3.6, we illustrate for $l = 2$ that there are $2^2! = 24$ possibilities to assign the four genotypes $\{a^g = 00, b^g = 01, c^g = 10, d^g = 11\}$ to the four phenotypes $\{a^p = 00, b^p = 01, c^p = 10, d^p = 11\}$. Each of the 24 genotype-

phenotype mappings represents a specific representation that assigns the fitness values to the genotypes in a different way and that results in a different difficulty of the problem.

In the following paragraphs, we investigate how problem difficulty changes for the one-max and deceptive trap problem if we use low-locality representations. The fitness function f_p for the fully easy l-bit one-max problem is defined as

$$f_p(x^p) = \sum_{i=0}^{l-1} x_i^p, \tag{3.24}$$

and the l-bit deceptive trap function is defined as:

$$f_p(x^p) = \begin{cases} l - 1 - \sum_{i=0}^{l-1} x_i^p & \text{for } \sum_{i=0}^{l-1} x_i^p < l, \\ l & \text{for } \sum_{i=0}^{l-1} x_i^p = l. \end{cases} \tag{3.25}$$

If we use genotypes and phenotypes of length l the number of possible representations is $2^l!$. To reduce this number, we assume without loss of generality that the phenotype x^p with only ones, which has fitness $f_p = l$, is always assigned to the individual x^g with only ones. Then, the number of different representations is reduced to $(2^l - 1)!$. For example, in Figs. 3.18 and 3.19 we have $2^l = 8$ genotypes and $2^l = 8$ phenotypes. Therefore, we have $8! = 40,340$ different representations. If we assign $x^g = 111$ always to $x^p = 111$ then there are only $7! = 5,040$ different representations. Every representation represents a different genotype-phenotype mapping.

Furthermore, we have seen in the previous subsection that for the used phenotype-fitness mapping f_p (one-max and deceptive trap problem) there are some genotype-phenotype mappings that do not modify the BBs and therefore do not change problem difficulty (both individuals have the same number of ones from item 1). These mappings have the same properties and the different f_g differ only for individuals that phenotypically have the same number of ones. There are $\prod_{i=1}^{l} \binom{l}{i}!$ representations of that kind which we want to denote as "high-locality equivalent". For example, in Fig. 3.18 we can change f_g and assign $x^g = 001$ to $x^p = 010$ and $x^g = 010$ to $x^p = 001$. Although we use a different representation, the assignment of the fitness values to the genotypes has not changed. This effect is a result of the redundancy of the used one-max and deceptive trap problem which both only consider the number of ones in the phenotype.

If we use these results we can calculate how many groups of different genotype-phenotype mappings exist that result in a different structure of the BBs. If we have $(2^l - 1)!$ different genotype-phenotype mappings and use a l-bit one-max or deceptive problem, then there are $\prod_{i=1}^{l} \binom{l}{i}!$ mappings that do not change the structure of the BBs and are equivalent to each other. Therefore, we have

$$\frac{(2^l - 1)!}{\prod_{i=1}^{l} \binom{l}{i}!}$$

groups of different genotype-fitness mappings. Each group consists of $\prod_{i=1}^{l} \binom{l}{i}!$ different genotype-phenotype mappings which do not affect the structure of the BBs and where only the mapping is changed between genotypes and phenotypes that have the same number of ones.

When using a 3-bit one-max or deceptive trap problem then there are $\frac{(2^3-1)!}{3!3!} = 5040/36 = 140$ groups of different genotype-fitness mappings with different properties. Each of these 140 different groups result in a different structure of the genotypic and phenotypic BBs. We use for our investigation 10 concatenated one-max or deceptive trap problems of size 3. Therefore, the overall string length $l = 30$, and the fitness of an individual is calculated as the sum over the fitness of the ten one-max or deceptive trap sub-problems.

To illustrate how genotype-phenotype mappings change the complexity of a problem we measure how many of the ten BBs (a BB is a correctly solved sub-problem and consists of a sequence of $l = 3$ ones) a GA finds dependent on the used representation. As we use a 3 bit problem there are 5,040 different representations and 5,040/36=140 different representations that are equivalent to each other. Only these representations that are contained in exactly one out of the 140 equivalence groups are "high-locality equivalent" ($d_m = 0$). An example is shown in Fig. 3.18. Due to the structure of the one-max and deceptive trap problem, there are 35 other representations which, although they have low locality, do not change the structure of the BBs and are "high-locality equivalent". These types of encodings assign genotypes and phenotypes with the same number of ones in a different way.

Figure 3.21 presents the results of our experiments for the one-max problem. We show the distribution of the number of correctly solved sub-problems at the end of a GA run when using different representations. The plot shows results for all 5,040 different representations. The ordinate counts the number of representations that allow a GA to correctly solve a certain number of sub-problems. We used a generational GA with tournament selection without replacement of size 2, uniform crossover, no mutation and a population size of $N = 15$. We performed 200 runs for each representation, and each run was stopped after the population was fully converged. The average number of correctly solved sub-problems measures the problem difficulty for the GA using one specific representation. The more sub-problems which could be correctly solved, the easier the problem is for GAs.

How can we interpret the data in Fig. 3.21? Every bar indicates the number of different representations that allow a GA to correctly identify a specific number of sub-problems. For example, the bar of height 95 at position 7.0 means that a GA correctly solves on average between 6.975 and 7.025 sub-problems for 95 different representations. The bar at position 4.85 means that there are 4 different representations that allow a GA to correctly solve on average between 4.825 and 4.875 sub-problems. The plot shows that by using a GA with only 15 individuals we solve independently of the used representation at least 4.2 sub-problems, and we are not able to correctly solve more

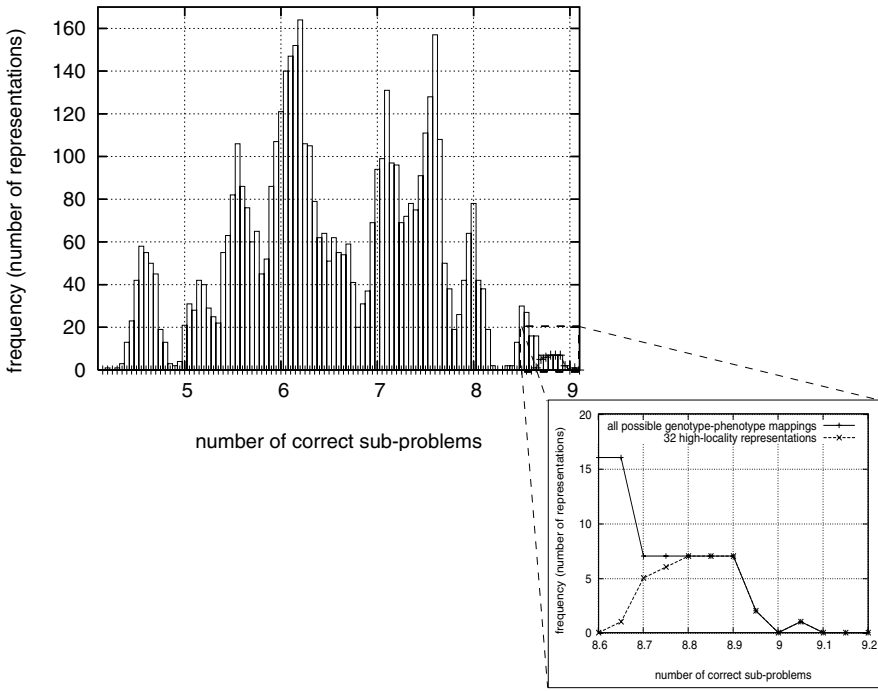


Figure 3.21. Experimental results of the frequency of the number of correct sub-problems at the end of a run for all possible encodings of a 3-bit one-max problem. We present results for 10 concatenated 3-bit problems. The optimal solution is always 111 so there are $(2^3 - 1)! = 5,040$ different representations. We use a GA with tournament selection without replacement, uniform crossover and a population size of $N = 15$. We perform 200 runs for every possible encoding. Only for these 36 representations that are equivalent to the high-locality representation, the fully easy one-max problem remains fully easy. All other encodings have low locality and make the problem more difficult to solve for GAs.

than 9 out of ten sub-problems. Furthermore, it is surprising that we have no normal distribution over the number of correct sub-problems but that there are clusters. For example, there are many representations that allow a GA to solve on average between 5.8 and 6.3 sub-problems but there are only a few representations that allow a GA to correctly solve on average between 6.5 and 6.8 sub-problems. The reason is, that there are only 140 different equivalence groups of representations. Although we have 5,040 different genotype-phenotype mappings, there are only 140 different levels of problem complexity possible. The observed clusters are probably a result of these small number of different levels of problem complexity.

It is more interesting to ask how the high-locality representation from Fig. 3.18 performs? And how the performance is of the other 35 genotype-

phenotype mappings that, although they have low locality and do not preserve the distances, are equivalent to the high-locality representation and result in the same genotype-fitness mapping? The small plot in Fig. 3.21 answers these questions. The bold line shows the performance of a GA using these 36 different "high-locality-equivalent" representations. The use of these representations results in the highest GA performance. For example, there are 7 different representations that allow a GA to correctly solve between 8.825 and 8.875 sub-problems. All 7 encodings belong to the group of 36 "high-locality-equivalent" encodings. Furthermore, we see that all representations that allow a GA to correctly solve on average more than 8.75 sub-problems belong to this group. These 36 encodings result in the highest proportion of correct sub-problems. For these encodings the one-max problem remains fully easy and the size of the genotypic and phenotypic BBs stays $k_g = k_p = 1$.

Changing the representation, that means assigning the elements of Φ_g in a different way to the elements in Φ_p, always results in a low-locality representation. If the genotype-fitness mapping is not "high-locality equivalent" then the BB-complexity increases and the problem becomes more difficult to solve. A GA has more difficulties in solving the problem, and the proportion of correctly solved sub-problems is lower. The plot illustrates nicely that the used representation can change the complexity of the one-max problem dramatically. However, only encodings that are equivalent to the high-locality encoding allow a GA to efficiently solve the fully easy one-max problem.

In Fig. 3.22, we present results for 10 concatenated instances of a 3-bit deceptive trap. The GA parameters chosen are the same as for the one-max problem. The plots show that, as expected, the GA performs worst for "high-locality equivalent" representations. All other representations make the problem easier to solve for GAs.

We have empirically shown that only high-locality representations guarantee that fully easy problems remain fully easy. High-locality representations preserve BB-complexity and are a good choice if we want GAs to reliably solve problems of bounded complexity. As soon as a representation has low locality and does not preserve BB-complexity, some of the easy problems become more difficult and therefore can no longer be solved by the GA. Indeed, some of the difficult problems could become solvable by using low-locality representations, but in general we are not interested in solving these types of problems.

3.3.8 Conclusions

This section presented the third and final element of a theory of representations. We investigated how the locality of a representation influences the performance of GEAs. The locality of a representation describes how well genotypic neighbors correspond to phenotypic neighbors. It is high if genotypic neighbors correspond to phenotypic neighbors. The results show that high-locality representations preserve the difficulty of a problem and phenotypically easy problems also remain genotypically easy. Using low-locality

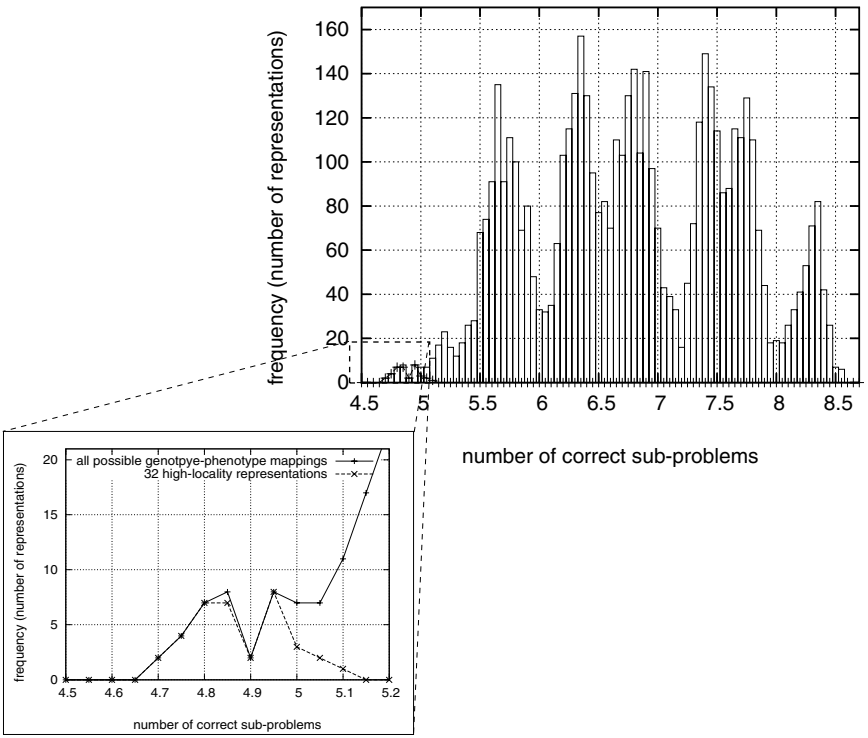


Figure 3.22. Experimental results of the frequency of the number of correctly solved sub-problems at the end of the run for all possible encodings of a 3-bit deceptive-max problem. We present results for 10 concatenated 3-bit problems. The optimal solution is always 111 so there are $(2^3 - 1)! = 5,040$ different possible encodings. The GA uses tournament selection without replacement, uniform crossover and a population size of $N = 15$. We perform 200 runs for every possible encoding. Only for these 36 representations that are equivalent to the high-locality encoding does the fully difficult deceptive trap remain fully difficult. For all other representations, the BB-complexity is reduced and the problem becomes easier to solve for GAs.

representations is equivalent to randomizing the search process. Therefore, low-locality representations change problem difficulty and make easy problems more difficult and deceptive problems more easy to solve.

In general, we want GEAs to be able to solve a class of problems of bounded complexity fast and reliably. However, the results have shown that the use of low-locality representations in general changes problem difficulty and can only increase problem difficulty for fully easy problems. Therefore, easy problems that are solvable using a high-locality representation could become unsolvable when using a low-locality representation that modifies BB-complexity. To guarantee that a GA can reliably solve problems of bounded complexity it is designed for, we recommend the use of high-locality representations.

This section nicely illustrated that representations can dramatically change the complexity of a problem. The presented work has shown that even fully difficult problems can be solved easily if a proper, low-locality, representation is used. However, using the same representation for a fully easy one-max problem can make the problem fully difficult and unsolvable. Therefore, the use of low-locality representations could be advantageous if we know that the problem is deceptive. But in general, users do not have this information and therefore, they should not use these types of representations.

3.4 Summary and Conclusions

In Sect. 3.1 we described, analyzed, and modeled the effect of redundant representations on the performance of GEAs. We distinguished between synonymously and non-synonymously redundant representations and illustrated that non-synonymous redundancy does not allow genetic operators to work properly and therefore reduces the efficiency of evolutionary search. For synonymously redundant representations, GEA performance depends on the change of the initial supply. Based on this observation models were developed that give the necessary population size for solving a problem, and the number of generations as $O(2^{k_r}/r)$, where k_r is the order of redundancy and r is the number of genotypic BBs that represent the optimal phenotypic BB. As a result, uniformly redundant representations do not change the performance of GAs. Only by increasing r, which means overrepresenting the optimal solution, does GA performance increase. Therefore, non-uniformly redundant representations can only be used advantageously if there exists a-priori some information about the optimal solution.

This was followed in Sect. 3.2 by an investigation into how the scaling of an encoding influences the performance of GEAs. We extended previous work (Rudnick 1992; Thierens 1995; Thierens et al. 1998; Harik et al. 1997) and formulated a more exact convergence model considering genetic drift for exponentially scaled representations. Representations are exponentially scaled if the contribution of the genotypic alleles to the construction of the phenotypic alleles is exponentially different. Using the developed population sizing model, we were able to more accurately predict the behavior of GEAs using exponentially scaled representations.

Finally, we presented the third and final element of a theory of representations, namely the influence of locality on problem complexity. In Sect. 3.3, we showed that high-locality representations, which preserve the neighborhood structure when mapping genotypes to phenotypes, do not modify the difficulty of a problem. When using low-locality representations, on average problem difficulty changes. On average, fully easy problems become more difficult, and deceptive problems easier. We have discussed why representations that keep easy problems easy and make deceptive problems easier are nice

to have, but not possible without having an exact knowledge about the optimization problem a priori.

In this chapter, we identified three important elements towards a general theory of representations. We identified redundancy, scaling, and locality/distance distortion as having a major influence on the performance of GEAs. We were able to show that synonymously redundant encodings do not modify the performance of a GEA as long as the representation is uniformly redundant. Our investigation into non-uniformly scaled representations has shown that these types of encodings prolong the search process and increase the problems of GEAs with genetic drift. Finally, we have seen that low-locality representations do not preserve BB-complexity in general and make phenotypically easy problems more difficult. Therefore, to make sure that GAs are able to reliably solve easy problems and problems of bounded complexity, the use of high-locality representations is recommended.

Even by only presenting some basic elements of a general theory of representations we are able to analyze and predict the behavior and performance of GEAs using existing representations significantly better. The presented theory gives us a deeper understanding on how existing representations influence the performance of GEAs, as well as allows us to design new representations in a more theory-guided way. By using the presented theory, on the one hand we can develop general and robust representations that can be applied to problems of unknown complexity, and on the other hand problem-specific representations which could fail for some problems, but perform well for a specific problem.

Although the provided elements of representation theory already allow a guided design and analysis of representations, further research is still necessary to develop a general representation theory. Especially, the relationship between the presented elements of theory of representations should be investigated more deeply. We believe that as we are able to easily separate the effects of redundant and exponentially scaled representations that there is not much interconnection and overlapping between these two elements of theory. However, for locality and its influence on BB-complexity, the situation is different. We have seen that the modification of problem complexity is strongly influenced by redundancy or scaling. Therefore, further research is necessary to identify the exact relations between the presented elements of theory.

Finally, we want to encourage researchers to do more basic research towards the development of a general theory of representations. We believe that we provided some important parts, but there is still a long way to go. However, the path is worth following, as a general theory of representations would allow us to unleash the full power of genetic and evolutionary search and help us to solve problems fast, accurately and reliably.

4

Time-Quality Framework for a Theory-Based Analysis and Design of Representations

Over the last decades, researchers gained more and more knowledge about the principles of genetic and evolutionary algorithms (GEAs) and were able to formulate a theory describing the behavior of GEAs more precisely (Bäck et al. 1997; Vose 1999; Goldberg 2002; Reeves and Rowe 2003). The existing elements of GEA theory explain quite accurately the influence of many important GEA parameters, as well as selection, recombination, or mutation methods on the performance of GEAs. By using the existing GEA theory, straight forward design and the development of new, competent GAs (Goldberg et al. 1993; Harik and Goldberg 1996; Pelikan et al. 1999; Pelikan et al. 1999) became possible. However, concerning representations for GEAs, a framework which describes the influence of representations on the performance of GEAs is still missing, although it is well known that the used representation has a strong influence on GEA performance. Such a framework could help us to develop new representations in a more theory-guided manner and would be an important step towards a general theory of representations for GEAs.

The purpose of this chapter is to develop a framework for a theory-based analysis and design of representations for GEAs based on the elements of theory we presented in the previous chapter. The framework should allow us to model and predict the influence of different types of representations on the performance of genetic and evolutionary search. It should describe how redundancy, scaling, and locality of a representation influence the time to convergence and the expected solution quality. By using the framework, we would be able to theoretically compare the efficiency of different representations, as well as to design new representations in a theory-guided way.

The chapter starts with a brief overview of the determinants – time and quality – of GEA performance. In Sect. 4.2, the elements of the framework, namely redundancy, scaling, and locality/distance distortion, are presented. We review their influence on representations, formulate how the three properties of representations can be measured, and describe how genetic and evolutionary search is affected. In Sect. 4.3, the framework itself is described. We formulate how the probability of error α and the convergence time t_{conv}

depend on the different elements of the framework. Because we are not yet able to consider the effect of scaled representations in general, the section is split up into two parts concerning uniformly and non-uniformly scaled representations. For both types of scaled representations we describe how the solution quality and the time to convergence depends on the redundancy and locality of an encoding. This is followed in Sect. 4.4 by some implications of the framework on the design of representations. We show how the use of representations with different properties affects the supply of BBs, the dynamics of genetic search, or the size of BBs. The chapter ends in Sect. 4.5 with a summary and concluding remarks.

4.1 Solution Quality and Time to Convergence

The following section briefly reviews determinants for GEA performance. It focuses on solution quality and time to convergence.

For comparing the efficiency of different GEAs using different types of representations, a measurement of GEA performance is necessary. Widely used determinants for GEA performance are the solution quality and the time to convergence. In general, the solution quality and convergence time depend on the used genetic operators, the GEA parameters, the used representation, and the optimization problem itself.

The solution quality of GEAs can be measured by the probability P_n of GEA success. GEA success means that the optimal solution is found by the GEA. When using the more common probability of GEA failure α, GEA success is defined as $P_n = 1 - \alpha$. Earlier work by Harik et al. (1997) has shown that when using selectorecombinative GAs, the probability of error $\alpha = 1 - P_n$ goes with $O(e^{-N})$. With decreasing α the population size N increases exponentially. Therefore, instead of using α, the population size N that is necessary for solving a problem can also be used for comparing GEA performance. Measuring GEA performance becomes more complicated if the best solution is not known a priori. Then, P_n cannot be calculated and the best fitness at the end of a GEA run can be used. It corresponds to the probability of GEA success P_n, and is determined by the used population size N.

The time to convergence t_{conv} describes how many generations selectorecombinative GEAs need to converge completely. A population is converged if there is no genetic diversity in the population after t_{conv} generations and all individuals in the population represent the same phenotype. It was shown (Thierens and Goldberg 1993; Miller and Goldberg 1996b; Miller and Goldberg 1996a) that the convergence time mainly depends on the length of the string l and the used selection scheme. As soon as the population size N is large enough to solve the problem reliably, the convergence time t_{conv} does not depend on N any more.

To compare the overall performance of different GEAs the number of fitness evaluations n_f can be used. For a given solution quality $P_n \gg 0$ the total

number of fitness calls can be calculated as

$$n_f = N \times t_{conv}.$$

4.2 Elements of the Framework

We focus in this section on the properties of representations that influence GEA performance and describe the elements of representation theory that are used in the framework. We review the elements, describe how we can measure them, and illustrate their effects on GEAs. In Chap. 3, we presented redundancy, scaling, and locality/distance distortion as relevant properties of representations. Although we believe that these three elements are some of the most important elements of the time-quality framework, there could still be others. Finding and describing them is left to further research.

The section consists of three subsections which discuss the single elements, namely redundancy, scaling, and locality. In each subsection, we briefly describe what we mean, illustrate how redundancy, scaling, or locality can be measured, and finally describe how GEAs are affected.

4.2.1 Redundancy

Section 3.1 shows that the use of redundant encodings affects the performance of GEAs. In the context of representations, redundancy means that on average one phenotype is represented by more than one genotype. Therefore, $|\Phi_g| > |\Phi_p|$ when using redundant representations. Consequently, a representation is not redundant if $|\Phi_g| = |\Phi_p|$. Then, the number of genotypes is the same as the number of phenotypes. Because we assume that every phenotype must be represented by at least one genotype, the size of the genotypic space can not be smaller than the size of the phenotypic space.

To model the effects of redundancy, we distinguish between synonymous and non-synonymous redundancy. When focusing on synonymous redundancy, the order k_r of redundancy is introduced (see Sect. 3.1.2). It measures the amount of redundant information in the encoding (in bit). There are k_r bits and 2^{k_r} different possibilities (individuals) to encode 1 Bit of information content (2 possibilities). Using no redundancy in an encoding results in $k_r = 1$. Furthermore, r is defined as the number of genotypic BBs of size kk_r that represent the optimal phenotypic BB of size k. Therefore, for non-redundant encodings $k_r = 1$ and $r = 1$. We know from (3.2) that for redundant encodings

$$r \in \{1, 2, \ldots, 2^{kk_r} - 2^k + 1\}.$$

We assume that we use binary strings ($\chi = 2$). For binary genotypes, there are 2^k different phenotypes and they are represented by 2^{kk_r} different genotypes. For non-binary genotypes we refer to Sects. 6.4.4 and 6.5.5. Using uniformly redundant representations results in

$$r_{uniform} = 2^{k(k_r-1)}$$

and $x_0/N = r/2^{kk_r} = 1/2^k$. x_0 denotes the the initial supply of BBs. Therefore, a representation is non-uniformly redundant if $r/2^{kk_r} \neq 1/2^k$. For $r/2^{kk_r} > 1/2^k$ the optimal solution is overrepresented, and for $r/2^{kk_r} < 1/2^k$ the optimum is underrepresented.

Our investigation into the effects of redundancy on GEAs in Sect. 3.1.3 has shown that the supply of BBs in the initial population is influenced by the use of non-uniformly redundant encodings. If the optimal solution is overrepresented by the used synonymously redundant representation the performance of GEAs is increased, that means lower run duration t_{conv} and lower probability of error α. The situation is reversed if the optimal solution is underrepresented and t_{conv} and α increases.

4.2.2 Scaling

In Sect. 3.2, we discussed the effects of exponentially scaled representations on the performance of GEAs. Representations are uniformly scaled, if all genotypic alleles have the same contribution to the construction of the phenotypic alleles. Therefore, GEAs using uniformly scaled representations solve all alleles implicitly in parallel. In contrast, a representation is non-uniformly scaled if some genotypic alleles have a higher contribution to the construction of the phenotypic alleles than others. As a result, domino convergence occurs and the alleles are solved sequentially according to their salience. The most salient alleles are solved first, whereas the lowest salient alleles are solved last.

To more formally describe the scaling of a representation, a measurement of how strong a representation is scaled is necessary. Therefore, we describe by the order of scaling s the difference in salience for the different alleles. When using a binary genotype of length l and ordering the alleles according to their contribution to the fitness in ascending order, we define the order of scaling $s \in [1, \infty[$ as

$$s = \frac{1}{l-1} \sum_{i=1}^{l-1} \frac{x_{i+1}^c}{x_i^c},$$

where x_i^c denotes the contribution of the ith most salient allele to the phenotype, and $x_{i+1}^c \geq x_i^c$, for $i \in \{1, \ldots, l-1\}$. Therefore, x_1^c denotes the contribution of the lowest salient allele and x_l^c denotes the contribution of the most salient allele. When using uniformly redundant encodings the contribution of all alleles is the same which results in $x_i^c = \text{const}$, for $i \in \{1, \ldots, l\}$. Therefore, $s = 1$ for uniformly scaled representations. When using exponentially scaled representations the order of scaling $s > 1$ is constant; for binary encoded strings, we get $s = 2$.

The order of scaling s influences the dynamics of genetic search. With increasing s, the alleles are solved more and more sequentially. Rudnick (1992) proposed the use of a convergence window for modeling the dynamic solving

process when using non-uniformly scaled representations. The convergence window is a set of contiguous alleles that are not yet fully converged but have started to converge. The size $\lambda_c \in \{1, \ldots, l\}$ of the convergence window is equal to l for uniformly scaled encodings and equal to one for $s \to \infty$. $\lambda_c = 1$ results in strictly sequential solving of the alleles, whereas for $\lambda_c = l$ all alleles are solved in parallel.

With increasing order of scaling s, the size λ_c of the convergence window is reduced. Earlier work (Thierens et al. 1998; Lobo et al. 2000) shows in correspondence to the results of Sect. 3.2.4, that the assumption of a convergence window of size $\lambda_c = 1$ results for $s = 2$ (exponentially scaled representations) in a good approximation of the dynamics of GEA search. However, for a more general theory of scaled representations a more detailed analysis of the interdependencies between λ_c and s is necessary.

4.2.3 Locality

In Sect. 3.3, we saw that when using a representation, the neighborhood structure can be different for the genotypes and phenotypes. In this case, the size and length of the genotypic and phenotypic building blocks are different.

We have illustrated that high-locality representations guarantee that the complexity of a problem is preserved and easy problems remain easy. Therefore, high locality is necessary for efficient mutation-based search, low distance distortion is necessary for efficient crossover-based search, and high locality is a necessary condition for low distance distortion. If a representation has low locality, some genotypic neighbors are not phenotypic neighbors, and the difficulty of the optimization problem is changed. Our investigation into the influence of locality on problem difficulty has shown that only high locality guarantees that the problem difficulty remains unchanged. We defined in (3.23) the locality d_m of a representation as

$$d_m = \sum_{d^g_{x,y} = d^g_{min}} |d^p_{x,y} - d^p_{min}|,$$

where $d^p_{x,y}$ is the phenotypic distance between the phenotypes x and y, $d^g_{x,y}$ is the genotypic distance between the corresponding genotypes, and d^p_{min}, respective d^g_{min} is the minimum distance between two (neighboring) phenotypes, and respectively genotypes. Without loss of generality we want to assume that $d^g_{min} = d^p_{min}$. For $d_m = 0$ the genotypic neighbors correspond to the phenotypic neighbors, the encoding has perfect locality, and the complexity of the phenotypic problem is not modified.

We have seen in Sect. 3.3 that by using low-locality representations ($d_m \neq 0$), the complexity of the problem which can be measured by the size k of the BBs, can be changed. Distinguishing between the size k_p of BBs in the phenotypic space, and the size k_g of BBs in the genotypic space allows us to model the influence of d_m on the performance of GEAs more exactly. Section

3.3 has shown that high-locality representations preserve problem difficulty and the problem has the same genotypic as phenotypic complexity:

$$k_g = k_p \text{ , if } d_m = 0.$$

The situation becomes more complicated if $d_m \neq 0$. As soon as neighboring genotypes do not correspond to neighboring phenotypes, the complexity of the BBs is changed and $k_g \neq k_p$. Every phenotypic problem with complexity k_p can be transformed by the use of a low-locality representation into a genotypic problem with complexity $k_g \in [1, l]$. For every problem there is always a representation that results in a fully easy problem, $k_g = 1$, as well as a representation that results in a fully difficult, misleading trap with $k_g = l$ (Liepins and Vose 1990). Therefore, when using low-locality representations ($d_m \neq 0$), the genotypic size of BBs k_g depends not only on the genotype-phenotype mapping f_g, but also on the specific structure of the phenotypic problem f_p. Therefore, we get

$$k_g = \begin{cases} k_p & \text{, for } d_m = 0, \\ k_g(f_g, f_p) \text{ , with } 1 \leq k_g \leq l & \text{, for } d_m \neq 0, \end{cases} \tag{4.1}$$

where l denotes the length of the binary string.

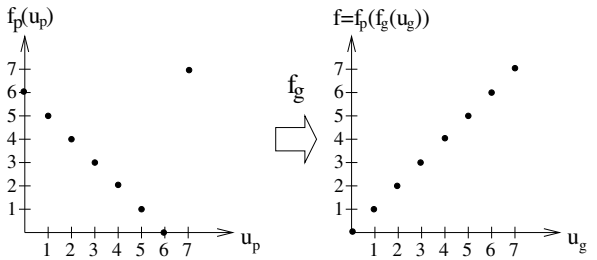
We want to illustrate with a small example why k_g does not only depend on f_g and k_p, but also on f_p. Section 3.3.1 has discussed that every fully deceptive trap with $k_p = l$ (3.25) can be transformed into a fully easy problem with $k_g = 1$ by a linear transformation. For this purpose we want to define the genotype-phenotype mapping f_g as

$$u_g = \begin{cases} l - u_p - 1 & \text{if } u_p \neq l, \\ l & \text{if } u_p = l, \end{cases} \tag{4.2}$$

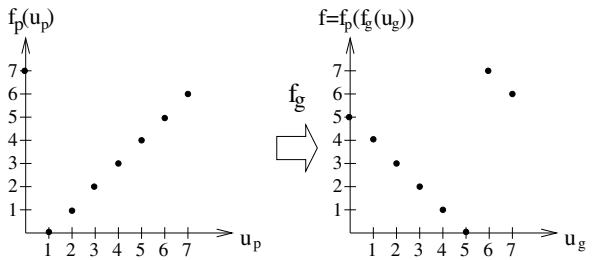
where l is the length of the string, u_g is the number of ones in the genotype and u_p is the number of ones in the phenotype. This encoding has low locality ($d_m \neq 0$). By the low-locality encoding f_g the phenotypically fully deceptive trap with $k_p = 7$ becomes fully easy (see Fig. 4.1(a)). However, using the same mapping f_g for a different phenotypic problem of the same complexity (Fig. 4.1(b)) does not significantly reduce k_g. As the size $k_p = k_g = 7$ of BBs remains unchanged, and only the position of the optimal solution is changed, the problem is still fully deceptive ($k_g = l$) after applying f_g. We see that for predicting k_g the knowledge of k_p is not enough when using low-locality representations. It is necessary to know f_p as well as f_g to predict the influence of low-locality representations on the difficulty of the problem.

4.3 The Framework

This section provides the time-quality framework modeling the influence of representations on the performance of genetic and evolutionary algorithms.



(a) The genotype-phenotype mapping f_g defined in (4.2) makes the fully deceptive phenotypic trap (left) fully easy (right).



(b) Because the optimal solution is located at a different position ($u_p = 0$) the genotype-phenotype mapping f_g from above does not reduce the complexity of the BBs significantly.

Figure 4.1. We show how a representation f_g (4.2) modifies the complexity of different phenotypic fully deceptive problems. If the optimal solution is located at $u = 7$ the problem becomes fully easy ($k_g = 1$). However, if the optimal solution is located at $u = 0$ the complexity of the problem remains approximately unchanged. We see that for designing a representation f_g that makes difficult problems easier, the structure of the phenotypic optimization problem f_p must be known. Therefore, theory-guided design of low-locality representations that reduce BB-complexity is difficult and problem-specific.

The framework allows us to theoretically predict and compare the performance of GEAs using different types of representations. Therefore, thorough analysis and theory-guided design of representations becomes possible. Although the presented framework is not yet complete and there are still some gaps, rough approximations, unclear interdependencies, and also more as yet unknown elements, we believe that the framework is an important step towards a more general theory of representations.

The framework itself is based on the characteristics of the used encoding we introduced in Sect. 4.2. There, we have seen that the redundancy of a representation can be described by the order of redundancy k_r and the number of copies r which are given to the optimal solution. Furthermore, the modification of BB-complexity is determined by the locality d_m, which measures how well phenotypic neighbors correspond to the genotypic neighbors. Finally,

the scaling of a representation can be described by using the order of scaling s. Currently there is no general model available for the influence of s on the performance of GEAs. Therefore, we want to focus in this framework on uniformly scaled representations ($s = 1$) and exponentially scaled representations with $s \geq 2$.

The structure of the section follows the still missing general model of the influence of scaling. Therefore, the section is split into two parts. In Sect. 4.3.1, we present the part of the framework for uniformly scaled representations and in Sect. 4.3.2 we focus on exponentially scaled representations.

4.3.1 Uniformly Scaled Representations

We present the part of the framework that describes the influence of representations on GEA performance if the representations are uniformly scaled. We describe how the probability of error α and the time to convergence t_{conv} depend on redundancy and locality.

Based on the work from Harik et al. (1997) we get for the probability of error

$$\alpha = 1 - \frac{1 - (q/p)^{x_0}}{1 - (q/p)^N},$$

where x_0 is the expected number of copies of the best BB in the randomly initialized population, $q = 1-p$ is the probability of making the wrong decision between two competing BBs, and N is the population size. From (3.6) we know that

$$x_0 = N\frac{r}{2^{kk_r}},$$

where k is the phenotypic size of BBs, r is the number of genotypic BBs of length kk_r that represent the best phenotypic BB, and k_r is the order of redundancy. After some approximations (see Sect. 3.1.4) we finally model in (3.9) the influence of redundant encodings on the population size N as

$$N = -\frac{2^{k_r k-1}}{r} \ln(\alpha) \frac{\sigma_{BB}\sqrt{\pi m'}}{d}, \tag{4.3}$$

where $m' = m - 1$ with m is the number of BBs, d is the signal difference, and σ^2_{BB} is the variance of the BBs. The probability α of GA failure can be calculated as:

$$\alpha = \exp\left(-\frac{Ndr}{2^{k_r k-1}\sigma_{BB}\sqrt{\pi m'}}\right) \tag{4.4}$$

We have described in Sect. 4.2.3 that the problem difficulty measured by the size of BBs k is modified by the locality d_m of the representation. In (4.1), the genotypic size k_g of the BBs is calculated as

$$k_g = \begin{cases} k_p & \text{, if } d_m = 0, \\ k_g(f_g, f_p) \text{ ,where } 1 \leq k_g \leq l & \text{, if } d_m \neq 0, \end{cases}$$

where f_g is the genotype-phenotype mapping (the used representation), and f_p is the optimization problem with the size k_p of the phenotypic BBs. Substituting k_g into 4.4 we get for uniformly scaled representations:

$$\alpha = \exp\left(-\frac{Ndr}{2^{k_r k_g - 1}\sigma_{BB}\sqrt{\pi m'}}\right). \tag{4.5}$$

The probability of error α goes with $O\left(\exp\left(\frac{-r}{2^{k_r k_g}}\right)\right)$. We see that using redundant representations ($k_r > 1$) without increasing r has the same influence on GEA performance as increasing the size of BBs k_g.

From Sect. 3.1.5 we get for the time to convergence for a uniformly scaled representation (3.10)

$$t_{conv} = \frac{\sqrt{l}}{I}\left(\frac{\pi}{2} - \arcsin\left(2\frac{x_0}{N} - 1\right)\right), \tag{4.6}$$

where l is the length of the phenotypes, and I is the selection intensity. Substituting x_0 from (3.6) into (4.6) yields

$$t_{conv} = \frac{\sqrt{l}}{I}\left(\frac{\pi}{2} - \arcsin(\frac{r}{2^{k_r k-1}} - 1)\right).$$

When considering the effect of locality (4.1) we finally get for the time to convergence

$$t_{conv} = \frac{\sqrt{l}}{I}\left(\frac{\pi}{2} - \arcsin(\frac{r}{2^{k_r k_g - 1}} - 1)\right). \tag{4.7}$$

t_{conv} increases with larger k_g and decreasing $r/2^{k_r}$. With $0 < \frac{r}{2^{k_r k_g}} < 1$ we can calculate upper and lower bounds for the expected time to convergence as

$$0 < t_{conv} < \frac{\sqrt{l}}{I}\pi$$

If $r/2^{k_r k_g} \approx 1$ most of the randomly created genotypic individuals represent the phenotypic optimum. Therefore, GEAs converge very fast and $t_{conv} \to 0$. If either k_g is a large number or $r/2^{k_r}$ is small then there is only a small fraction of optimal BB in the initial population and GEAs need many generations to converge.

4.3.2 Exponentially Scaled Representations

We describe the influence of redundancy and locality on the performance of GEAs if the representations are exponentially scaled. In contrast to the previous subsection where the size of the convergence window λ_c is equal to the string length and all alleles are solved in parallel, we assume that the alleles are solved strictly in serial and $\lambda_c = 1$.

As illustrated in Sects. 3.2 and 4.2.2 we can use the domino convergence model for estimating the performance of GEAs using exponentially scaled representations. We assume that the alleles are solved strictly in serial and there are no interdependencies between the l_s alleles in an exponentially scaled BB. However, it is possible to concatenate m exponentially scaled BBs of length l_s. When using exponentially scaled representations the maximum size of BBs is $k = 1$. All schemata of order $k = 1$ that contain the best solution have higher fitness than their competitors. Therefore, it makes no sense to consider the effect of locality on GEA performance when using exponentially scaled representations. Section 4.2.3 has shown that low locality modifies the size of BBs k and results in interdependencies between the alleles. However, if $k_g > 1$, the domino convergence model can not be used any more, because we can then not assume that the alleles are still solved sequentially. Therefore, we assume in the following that $k = 1$ and the representation does not modify the size of BBs when mapping the phenotypes onto the genotypes.

When using redundant representations we know from (3.6) that

$$\frac{x_0}{N} = \frac{r}{2^{kk_r}},$$

where x_0 is the expected number of copies of the best BBs in the initial population, N is the population size, k is the size of BBs, $m' = m - 1$ with m is the number of BBs, k_r is the order of redundancy, and r is the number of genotypic BBs of size kk_r that represent the best phenotypic BB.

As we have seen in Sect. 3.2.2, the probability p of making the right choice between a single sample of each BB remains constant for the l_s bits in the exponentially scaled BB if we assume that all alleles which are not yet touched by the solving process remain in their initial state. Substituting x_0/N into (3.13), we get

$$p = \mathbb{N}\left(\frac{1}{\sqrt{2\frac{r}{2^{k_r}}\left(1 - \frac{r}{2^{k_r}}\right)\left(\frac{4}{3}m - 1\right)}}\right). \tag{4.8}$$

As illustrated above, $k = 1$ and there are m competing BBs with l_s exponentially scaled alleles. Furthermore, with $x_0 = \frac{Nr}{2^{k_r}}$ we get from (3.15) for the probability of error

$$\alpha = \frac{(1/p-1)^{x_0} - (1/p-1)^N}{1 - (1/p-1)^N} = \frac{(1/p-1)^{\frac{Nr}{2^{k_r}}} - (1/p-1)^N}{1 - (1/p-1)^N}. \tag{4.9}$$

We want to approximate (4.9) in analogy to Sect. 3.1.4. If we assume that x_0 is small we get from (4.9)

$$\alpha \approx \left(\frac{1-p}{p}\right)^{x_0}.$$

When using the first two terms of the power series expansion of the normal distribution for approximating (4.8) we get

$$\alpha \approx \exp\left(x_0 \ln\left(\frac{1-x}{1+x}\right)\right),$$

where $x = 1/\sqrt{\pi \frac{x_0}{N}(1-\frac{x_0}{N})(\frac{4}{3}m-1)}$. Because x is a small number we can assume that $\ln(1-x) \approx -x$ and $\ln(1+x) \approx x$. Using these approximations we get

$$\alpha \approx \exp\left(-x_0 \frac{2}{\sqrt{\pi \frac{x_0}{N}(1-\frac{x_0}{N})(\frac{4}{3}m-1)}}\right).$$

If we approximate $\frac{x_0}{N}(1-x_0/N)$ by x_0/N we get for the probability of error

$$\alpha \approx \exp\left(-\frac{Nr}{2^{k_r-1}\sqrt{\pi \frac{r}{2^{k_r}}(\frac{4}{3}m-1)}}\right).$$

Simplifying this equation yields finally

$$\alpha \approx \exp\left(-\frac{2n\sqrt{r}}{\sqrt{2^{k_r}\pi(\frac{4}{3}m-1)}}\right). \tag{4.10}$$

Using this rough approximation we appreciate that α is reduced with increasing $r/2^{k_r}$ and N. α is also reduced with a smaller number m of competing BBs. The reader should notice that α does not depend on the length l_s of an exponentially scaled BB, as we assumed that the alleles remain in their initial state as long as they are not reached by the search window.

We have seen in Sect. 3.2 that genetic drift reduces the performance of GEAs when using exponentially scaled representations. Genetic drift can be considered by either the approximated drift model (3.21) or the stair-case drift model (3.22). By substituting the probability of error α either from (4.9) or from (4.10) into either (3.21) or (3.22) we get the average percentage of incorrect alleles $\bar{\alpha}$. For example, we can calculate the overall percentage $\bar{\alpha}$ of incorrect alleles using the approximated drift model as:

$$\bar{\alpha} = \frac{1}{l_s}\sum_{\lambda=0}^{l_s-1}\left(\left(1-s'(\lambda\frac{\pi}{2}\sqrt{\pi m})\right)\alpha + \frac{1}{2}s'(\lambda\frac{\pi}{2}\sqrt{\pi m})\right), \tag{4.11}$$

with

$$s'(t) = \begin{cases} 0 & \text{for } t < -N\ln(2/3), \\ 1-\frac{3}{2}\exp(-t/N) & \text{for } t > -N\ln(2/3). \end{cases}$$

With increasing l_s, more and more of the lower salient alleles are fixed randomly and $\bar{\alpha}$ is reduced.

The time to convergence for the m alleles of the same salience can be calculated by using (3.10) as

$$t_{conv} = \frac{\sqrt{m}}{I}\left(\frac{\pi}{2} - \arcsin\left(\frac{r}{2^{k_r-1}} - 1\right)\right). \tag{4.12}$$

As before we assume that $k = 1$. After m alleles of the same salience are converged the GEAs tries to solve the next m alleles with the next lower salience. Because each of the m BBs consists of l_s alleles with different salience and the solving process is strictly serial, we get for the overall time to convergence

$$t_{conv} = l_s\frac{\sqrt{m}}{I}\left(\frac{\pi}{2} - \arcsin\left(\frac{r}{2^{k_r-1}} - 1\right)\right), \tag{4.13}$$

The time to convergence increases linearly with the length of an exponentially scaled BB l_s. With larger $r/2^{k_r}$ the time to convergence is reduced.

4.4 Implications for the Design of Representations

The purpose of this section is to describe some of the important implications of the framework on the behavior of GEAs. We show how the influence of different types of representations on the performance of GEAs can be described by using the presented framework. Based on the framework, we see that representations that overrepresent a specific solution can result in high GEA performance, but are not robust concerning the location of the optimal solution. When using exponentially scaled representations, the framework tells us that there is a trade-off between the accuracy of the solution quality and convergence time. Because low-locality representations affect the size of BBs, the behavior of GEAs using low-locality representations is difficult to predict.

The section starts by illustrating the effects of non-uniformly redundant representations. We have seen in Sects. 3.1 and 4.2.1 that redundancy affects the supply of BBs in the initial population. Therefore, representations that overrepresent a specific solution result in high GEA performance but are not robust. Section 4.4.2 illustrates that the scaling of a representation influences the dynamics of genetic search. GEAs using exponentially scaled representations deliver rough approximations of the optimal solution after a few generations, but the overall time to convergence is increased in comparison to uniformly scaled representations. Finally, we show in Sect. 4.4.3 the effects of low-locality representations. If $d_m \neq 0$, the genotypic problem complexity depends on f_g and f_p and easy problems f_p on average become more difficult to solve for GEAs. Therefore, when using low-locality representations the performance of GEAs for a specific problem is difficult to predict but on average low-locality representations make easy problems more difficult.

4.4.1 Uniformly Redundant Representations Are Robust

Section 3.1 has illustrated the effects of redundancy on the performance of GEAs. The results have shown that the quality of the solutions and the time

to find them can be increased if we focus the genetic search on some specific areas of the search space.

We described in Sect. 4.2.2 the influence of synonymous redundancy by r denoting the number of genotypic BBs that represent the optimal phenotypic BB and k_r denoting the order of redundancy. Therefore, $r/2^{k_r}$ can be used for characterizing redundancy in an encoding.

Our framework in the previous section tells us how the solution quality (α and t_{conv}) depends on $r/2^{k_r}$. For uniform redundant representations (see (4.5) and (4.7))

$$\alpha = \exp\left(-\frac{Ndr}{2^{k_r k_g - 1}\sigma_{BB}\sqrt{\pi m'}}\right),$$

and

$$t_{conv} = \frac{\sqrt{l}}{I}\left(\frac{\pi}{2} - \arcsin\left(\frac{2r}{2^{k_r k_g}} - 1\right)\right).$$

When neglecting the effect of genetic drift we get for exponentially scaled representations (see (4.10) and (4.13))

$$\alpha \approx \exp\left(-\frac{2N\sqrt{r}}{\sqrt{2^{k_r}\pi(\frac{4}{3}m - 1)}}\right),$$

and

$$t_{conv} = l_s\frac{\sqrt{m}}{I}\left(\frac{\pi}{2} - \arcsin\left(\frac{r}{2^{k_r - 1}} - 1\right)\right)$$

We see that α goes for uniformly scaled representations with $O(\exp(-r/2^{k_r}))$ and for exponentially scaled representations with $O(\exp(-\sqrt{r/2^{k_r}}))$. The time to convergence t_{conv} is reduced for both types of representations with increasing $r/2^{k_r}$. Therefore, GEA performance increases with larger $r/2^{k_r}$. As a result designing efficient representations seems to be quite an easy task. Initially it appears that we simply have to increase $r/2^{k_r}$ and are rewarded with high performing GEAs. Therefore, we have to investigate if there are any problems associated with increasing $r/2^{k_r}$.

When using synonymously redundant representations, the order of redundancy k_r does not depend on the structure of the optimal solution. However, r depends by definition on the structure of the optimal solution. r measures how many genotypic BBs of size kk_r represent the optimal phenotypic BB of size k. On average $r_{avg} = 2^{k(k_r - 1)}$ genotypic BBs represent one of the 2^k phenotypic BBs. Therefore, if $r > r_{avg}$ for some phenotypic individuals, there must also be some individuals with $r < r_{avg}$. That means if some individuals are overrepresented by a specific representation there must be others which are underrepresented.

We have learned from the framework that solution quality increases with increasing $r/2^{k_r}$. If we have uniform redundancy ($r = 2^{k(k_r - 1)}$), GEAs perform the same as without redundancy. For uniformly redundant representations, the performance of GEAs is independent of the location of the optimal

solution. If a specific phenotype x_r^p is overrepresented with $r > 2^{k(k_r-1)}$, GEAs searching for optimal solutions that are similar to x_r perform better. However, when using this representation and searching for solutions that have a large distance to x_r^p, GEAs perform worse. The situation is vice versa if x_r^p is underrepresented. We see that by increasing $r/2k_r$, we reduce the robustness of the representation. A representation is denoted to be robust if the performance of a GA is independent of the location of the optimal solution in the search space. We illustrate this behavior of redundant encodings in Fig. 4.2. The Figure shows how the performance of GEAs depends on the over- or underrepresentation of the phenotype x_r^p.

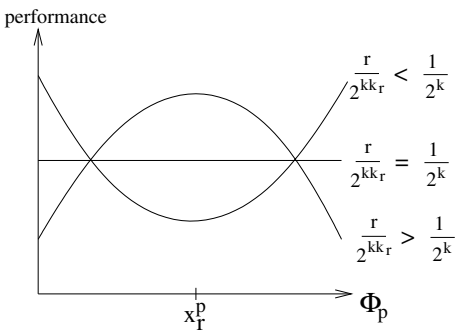


Figure 4.2. We show how the performance of GEAs using redundant representations depends on the location of a specific individual x_r^p in the search space Φ_p. r determines the number of genotypes that represent a specific phenotype x_r^p. The performance of GEAs is independent of r if all phenotypes are uniformly represented ($r = 2^{k(k_r-1)}$ for all phenotypes). If x_r^p is overrepresented ($r > 2^{k(k_r-1)}$) GEAs perform better when searching for individuals similar to x_r^p, and worse for individuals with a larger distance to x_r^p. If x_r^p is underrepresented the situation is reversed.

We see that when designing representations, redundancy is helpful if the number of copies r that are given to the optimal solution x_r^p is above average. However, to systematically increase the number of copies of the optimal solution, it is necessary to know where the optimal solution is located in the search space Φ_p. Otherwise, if we do not know where the optimal solution can be found, it is not possible to increase the number of copies of the best solution or solutions that are similar to the best solution in a systematic way by using redundant representations. Therefore, problem-specific knowledge is necessary to increase the value of r. If we do not have any problem-specific knowledge about the structure of the problem, either non-redundant, or uniformly redundant representations, should be used. Both types of encodings guarantee that GEAs perform robustly, that means independently of the structure of the optimal solution.

4.4.2 Exponentially Scaled Representations Are Fast, but Inaccurate

In Sect. 4.3, we examined the effect of uniformly and non-uniformly scaled representations on the performance of GEAs. We saw (see also Sect. 4.2.2) that a different scaling of representations modifies the dynamics of genetic search. When using uniformly redundant representations all alleles are solved in parallel, whereas for exponentially scaled BBs the alleles are solved strictly serially. In the following paragraphs, we want to illustrate that GEAs using exponentially scaled representations deliver fast solutions which are inaccurate.

In our framework, we have presented two different models for scaled representations. For uniformly scaled representations, we assumed that all alleles are solved in parallel, and that the size of the convergence window is the same as the string length. We get from (4.7) for $l = l_s m$ and $k_g = 1$

$$t_{conv}^{uniform} = \frac{\sqrt{l_s m}}{I} \left(\frac{\pi}{2} - \arcsin(\frac{r}{2^{k_r} - 1} - 1) \right) .$$

When using exponentially scaled representations, we use the domino convergence model and the size of the convergence window $\lambda_c = 1$. Therefore, we get from (4.13) for the overall time to convergence

$$t_{conv}^{exp} = l_s \frac{\sqrt{m}}{I} \left(\frac{\pi}{2} - \arcsin \left(\frac{r}{2^{k_r - 1}} - 1 \right) \right) .$$

In each of the m exponentially scaled BBs of size l_s, the alleles are solved strictly sequentially.

We see that when using exponentially scaled representations, the first alleles are converged to the correct solution after a short time and we get a first rough approximation of the correct solution. Furthermore, the alleles are solved from the most salient to the least salient. When using exponentially scaled representations the most salient allele has the same contribution to the phenotype as all lower salient alleles together. Because the low salient alleles do not significantly change the phenotype (and the corresponding fitness), we get an acceptable approximation after a few generations.

The situation is different when examining the number of generations that are necessary until the whole string is converged. GEAs using uniformly scaled representations converge faster and find the optimal solution after $t_{conv}^{uniform} = t_{conv}^{exp} / \sqrt{l_s}$ generations. We can compare the different times to convergence

$$t^{exp} < t_{conv}^{uniform} < t_{conv}^{exp},$$

where $t^{exp} = t_{conv}^{exp} / l_s$ denotes the time after m alleles of the same salience are converged (moving the convergence window to the next lower salient allele). GEAs always need more time to completely converge when using exponentially scaled representations than when using uniformly scaled representations.

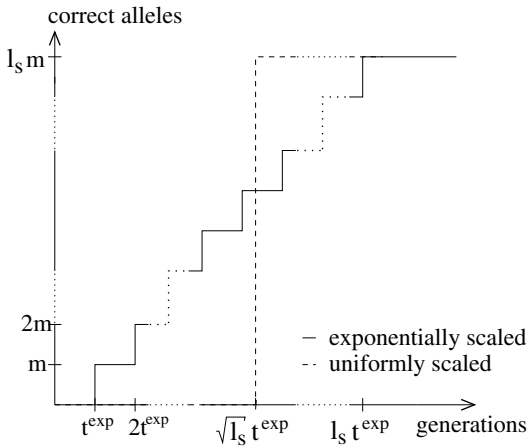


Figure 4.3. Number of correctly identified alleles over the number of generations for GEAs using uniformly versus non-uniformly scaled representations. The number of correctly identified alleles corresponds to the accuracy of the solution. GEAs using non-uniformly scaled representations provide an inaccurate solution to the problem more rapidly, but need longer to find the exact solution.

We want to illustrate the influence of scaling on the dynamics of genetic search in Fig. 4.3. The figure shows the number of correctly identified alleles over the number of generations using uniformly scaled versus non-uniformly scaled representations. The number of correctly identified alleles is a measurement for the accuracy of the solution we get. The plots show that GEAs using non-uniformly scaled representations steadily improve the solution quality. After a few generations GEAs already provide us with a correct, but yet inaccurate solution to the problem. GEAs using uniformly scaled representations do not give us approximations after a few generations, but allow us to find the exact optimum faster.

We see that non-uniformly scaled representations rapidly deliver correct, but inaccurate solutions, whereas uniformly scaled representations do not produce early results, but give us the optimal solution faster. If we do not want to spend much time and we are not interested in high accuracy, non-uniformly scaled representations are a considerable choice. On the other hand, if we need exact solutions, we should use uniformly scaled representations. GEAs using uniformly scaled representations can find the exact optimum in a shorter length of time.

4.4.3 Low-locality Representations Are Difficult to Predict, and No Good Choice

Section 3.3 illustrated how the complexity of an optimization problem can be modified by using low-locality representations. The genotypic size k_g of BBs is determined by the locality d_m (see Sect. 4.2.3). If the locality is high ($d_m = 0$) the genotypic problem complexity is the same as the phenotypic problem complexity. However, for $d_m \neq 0$ the genotypic size of BBs k_g depends not only on the used representation but also on the specific optimization problem.

We have already illustrated in Sect. 4.3.2 that when looking at exponentially scaled representations it makes no sense to consider the effect of d_m on

GEA performance. The performance of GEAs is modeled using the domino convergence model, which assumes strictly serial solving of the alleles. However, if the representation modifies the size of BBs and $k_g \neq 1$, there are interdependencies between the alleles, and the domino convergence model can not be used any more. Therefore, we want to focus in the following on uniformly scaled representations.

For uniformly scaled representations, the probability of error (4.5) can be approximated as

$$\alpha = \exp\left(-\frac{Ndr}{2^{k_r k_g - 1}\sigma_{BB}\sqrt{\pi m'}}\right),$$

and the time to convergence (4.7)

$$t_{conv} = \frac{\sqrt{l}}{I}\left(\frac{\pi}{2} - \arcsin(\frac{2r}{2^{k_r k_g}} - 1)\right),$$

where

$$k_g = \begin{cases} k_p & \text{, if } d_m = 0, \\ k_g(f_g, f_p) \text{ ,where } 1 \leq k_g \leq l & \text{, if } d_m \neq 0. \end{cases}$$

The probability of error goes with $O(\exp(1/2^{k_g}))$, and the time to convergence t_{conv} increases with increasing k_g. The genotypic size k_g of BBs is influenced by d_m. If $d_m = 0$, k_g is the same as k_p and the complexity of the problem is preserved when mapping the phenotypes onto the genotypes. The situation becomes different if $d_m \neq 0$. Then, k_g depends on the used representation f_g and on the optimization problem f_p. We have illustrated in the small example shown in Sect. 4.2.3 (see Fig. 4.1) that predicting GEA performance using low-locality representations is difficult. It depends on where the optimal solution is located, on the representation f_g, and the structure of the problem f_p.

We illustrate the problem with low-locality representations in Fig. 4.4. In Fig. 4.4(a), the locality is high and the size of BBs is the same in the genotypic and the phenotypic space. If we assume that the used GEAs can solve a problem up to $k_g = k_{max}$, we can be sure that all problems with $k_p < k_{max}$ can be solved reliably. The situation becomes different if $d_m \neq 0$ (Fig. 4.4(b)). Then, k_g depends not only on k_p, but also on f_g, and on f_p. Using a low-locality representation can result for problems with the same k_p in problems with different k_g. Therefore, we can not predict which types of problems remain solvable and which are no longer solvable. Originally easy problems could become fully difficult and fully difficult problems could become fully easy. To predict the performance of GEAs using low-locality representations is not possible if we have no detailed knowledge about f_p.

We summarize that we can not predict the performance of GEAs using low-locality representations if we have no exact knowledge about the optimization problem f_p and the used representation f_g. However, if we know that a problem is too difficult to be solved by the used GEAs, a representation with $d_m \neq 0$ can sometimes advantageously use this problem-specific information.

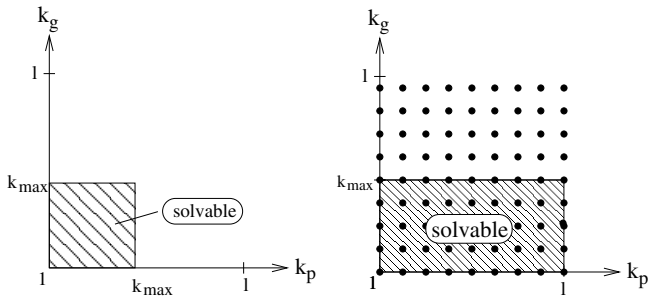


Figure 4.4. We illustrate how the genotypic size k_g of BBs depends on the phenotypic size k_p of BBs when using low-locality versus high-locality representations. For high-locality representations ($d_m = 0$), easy problems up to complexity k_p can be reliably solved (Fig. 4.4(a)). If $d_m \neq 0$ (Fig. 4.4(b)), k_g does not correspond to k_p, but depends on the used representation and the optimization problem. Then, we cannot predict k_g without knowing f_p.

Then, the representation modifies k_p and there is a chance that the problem becomes so easy that it can be solved ($k_g < k_{max}$). When a problem is fully difficult ($k_p = l$), on average, low-locality representations make the problem easier ($k_g \leq l$) and more likely to be solved by GEAs. If we have no information a priori about a problem, or if we know that a problem is easy to solve for GEAs, we strongly favor the use of high-locality representations. These types or representations allow GEAs to reliably solve easy problems up to some complexity bound.

4.5 Summary and Conclusions

We presented in this chapter a time-quality framework for a theory-based analysis and design of representations for genetic and evolutionary algorithms. The chapter started with the determinants of GEA performance. The performance of GEAs is determined by the expected quality of the solutions and the number of generations that are necessary to find them. This was followed in Sect. 4.2 by a description of the three elements the framework consists of. We presented how redundancy, scaling, and locality of a representation are measured and how they affect GEA performance. In Sect. 4.3, we presented the main part of the chapter: the framework. Based on the work outlined in Chap. 3, we showed how the probability of error α and the time to convergence t_{conv} is influenced by different properties of representations. Finally, we presented in Sect. 4.4 some implications of the framework on the design of representations.

Based on the three elements of representation theory outlined in Chap. 3, the presented framework theoretically describes how different types of representations influence GEA performance. The framework provides us with some important benefits. It gives us a theoretical model for a better understanding of the influence of representations on GEA performance. Furthermore, it allows us to model and predict the performance of GEAs using a specific representation for different types of optimization problems. Therefore, a theory-based use, analysis, and design of representations becomes possible by using the outlined framework.

Based on the results from Chap. 3, the framework shows that synonymously redundant representations increase GEA performance, if the optimal solution is overrepresented by the representation. However, if some specific individuals are overrepresented, others remain underrepresented, and the performance depends on the structure of the optimal solution. Only uniformly redundant representations are robust concerning the structure of the optimal solution.

By modifying the scaling of a representation the dynamics of genetic search are changed. If GEAs use exponentially scaled representations the domino convergence model can be used because the alleles are solved serially according to their salience. Therefore, the most salient alleles are solved after a few generations and a rough approximation of the optimal solution is available. However, to solve all alleles, GEAs using exponentially scaled representations need a larger number of generations compared to using uniformly scaled representations.

The presented framework reveals that the locality of a representation is crucial for the performance of GEAs. The locality describes how well genotypic neighbors correspond to phenotypic neighbors when mapping the phenotypes on the genotypes. The analysis shows that locality influences the genotypic size of BBs. When using high-locality representations, the complexity of the problem is preserved and easy problems remain easy. For low-locality encodings, the size of BBs is modified, and for predicting the resulting problem complexity, exact knowledge about the optimization problem and the used representation is necessary.

We believe that the representation framework developed in this chapter is an important step toward a more general theory of representations for GEAs. Although it is not yet completed and there are still many open questions and shortcomings, it provides us with a much better understanding of the principles of representations and allows a more theory-guided design of representations. We want to encourage researchers to use the presented framework as a basis for a more detailed and extensive investigation into representations.

5

Analysis of Binary Representations of Integers

In the previous chapter, we presented a framework which describes the effects of representations on the performance of GEAs. We illustrated the elements of the framework for problems where the phenotypes and genotypes are both bitstrings. The question is still open as to whether the framework also holds true for problems where the genotypes and phenotypes are different.

This question can be answered by examining problems where the genotypes are still binary but the phenotypes are integers. Integer optimization problems are common in many real-world applications. Although the most natural way for representing integer problems is to use an integer representation with integer genotypes, previous work has shown that by using genotypes that have a lower cardinality of the alphabet (for example binary genotypes) the possible number of schemata can be increased in comparison to integer strings (Goldberg 1990b). Consequently, researchers have developed different types of binary representations for integers. The most common are binary, Gray, and unary representations. Previous work has shown that these three representations have different properties and influence GEA performance differently (Caruana and Schaffer 1988; Whitley 1999; Whitley et al. 1997; Whitley 2000a).

The purpose of this chapter is to use the framework presented in the previous chapters to explain the performance differences of GEAs when using different binary representations for integers. For our investigation, we use two types of integer problems: variants of the easy integer one-max problem, and a difficult deceptive integer trap problem. For encoding integer phenotypes, we use binary, Gray, or unary representations.

The analysis of the unary encoding using the previously presented elements of representation theory reveals that the encoding is non-synonymously redundant and does not represent the phenotypes uniformly. Therefore, the performance of GEAs depends on the structure of the optimal solution. If the good solutions are overrepresented by the encoding, GEAs perform well, whereas, if the good solutions are underrepresented, GEAs fail.

The binary encoding uses exponentially scaled alleles to represent integer values. Therefore, the convergence behavior is affected by domino convergence and genetic drift. However, the analysis shows that genetic drift only results in a reduction of GEA performance for easy problems and small populations. An investigation into the locality of the binary encoding reveals that the locality is low and increases the difficulty of fully easy problems and reduces the difficulty of fully difficult problems.

Although the Gray encoding was designed to overcome the problems with the Hamming cliff (Schaffer et al. 1989), it also has low locality and changes the difficulty of problems. Focusing on selectorecombinative GAs, a schema analysis for the integer one-max problem reveals that using Gray encoding results in larger BBs in comparison to binary encoding. As a result, in comparison to the binary encoding, the difficulty of the easy integer one-max problem increases for selectorecombinative GAs. These results are not contradictory to the Free-Lunch theorem from Whitley (1999) and Whitley (2000a) regarding the Gray encoding but confirm the results therein. The difference can be found in the used search method. We investigate the influence of Gray encoding on recombination-based search approaches, whereas Whitley (1999) looks at mutation-based search methods. The work basically counts the number of local optima, which is lower when using Gray than binary encoding. Therefore, the performance of mutation-based search approaches on easy problems is higher when using Gray than when using binary encodings.

After a brief presentation of the integer problems in Sect. 5.1, Sect. 5.2 describes the Gray, binary and unary encodings and analyzes their properties. This is followed in Sect. 5.3 by a theoretical comparison of the three encodings using the elements of theory presented in Chap. 3. We illustrate how the unary encoding is affected by redundancy, how the exponential scaling of BBs influences the performance of the binary encoding, and how Gray encoding does not preserve problem difficulty well. Based on the elements of representation theory, we are able to make theoretical predictions about GEA performance. In Sect. 5.4, these predictions are finally confirmed by empirical results. The chapter ends with concluding remarks.

5.1 Integer Optimization Problems

In this section, we present integer problems we want to use for a comparison of different representations defined on binary genotypes.

To be able to make a fair comparison between different representations, the problem must be defined on the integer phenotypes independently of the used binary representation. The difficulty of the problem is determined by the phenotype-fitness mapping f_p. The difficulty of the problem can be changed by using an additional genotype-phenotype mapping f_g, which assigns binary genotypes to integer phenotypes. When assuming that the fitness function f_p assigns a real number to every individual in the phenotypic space, we get for

the phenotype-fitness mapping:

$$f_p(x^p) : \mathbb{N} \rightarrow \mathbb{R}.$$

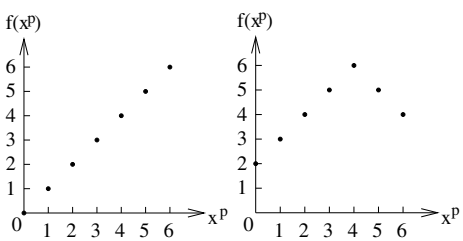
We want to use integer-specific variations of the one-max and the fully-deceptive trap problem. Traditionally, these problems are defined on binary strings, but we want to define them in a similar way for integers. The *integer one-max problem* is defined as

$$f_p(x^p) = x^p. \tag{5.1}$$

A more general variant of the integer one-max problem (denoted as *gen-one-max*) can be defined as:

$$f_p(x^p) = x_{max} - |x^p - a|, \tag{5.2}$$

where $x^p \in \mathbb{N}$, $x_{max} = \max(x^p)$, and $a \in \{0, 1, \ldots, x_{max}\}$. For $a = x_{max}$ the gen-one-max problem becomes the standard integer one-max problem (compare (5.1) and Fig. 5.1(a)). The difficulty of different gen-one-max problems is independent of the parameter a. a only changes the location of the optimal solution in the search space and evolutionary search algorithms should show the same performance for different values of a. Two examples for the gen-one-max problem are given in Fig. 5.1.



(a) $a = 6$ (b) $a = 4$

Figure 5.1. Two examples for the general integer one-max problem (gen-one-max)

Furthermore, the *integer deceptive trap problem* can be defined as

$$f(x^p) = \begin{cases} x^p & \text{if } x^p = x_{max}, \\ x_{max} - x^p - 1 & \text{else,} \end{cases} \tag{5.3}$$

where $x^p \in \mathbb{N}$. The one-max problems for integers are fully easy problems, whereas the integer deceptive trap should be fully difficult to solve for GEAs.

For measuring the similarity of individuals, we need to define a metric for the genotypic search space Φ_g and phenotypic search space Φ_p (see Sect. 3.3.2). As we use binary genotypes, we use the Hamming distance (Hamming 1980) on Φ_g, and the distance between two genotypes x^g and y^g of length l is defined

as $d_{x^g,y^g} = \sum_{i=0}^{l-1} |x_i^g - y_i^g|$. The distance measures the number of alleles that are different in both genotypes. The more bits two genotypes have in common, the more similar they are. The Hamming metric is chosen with respect to the bit-flipping operator. Using this mutation operator results in an individual that has the lowest possible genotypic distance from its parent. Following the Hamming metric for the genotypes, we measure the distance between two phenotypes x^p and y^p ($x^p, y^p \in \mathbb{N}$) as $d_{x^p,y^p} = |x^p - y^p|$. The distance between two phenotypes is simply the difference between both integers.

5.2 Binary String Representations

After we have defined the optimization problems, we present possible binary representations f_g for integers. The representation f_g assigns binary genotypes x^g to integer phenotypes x^p.

Instead of using binary strings with cardinality $\chi = 2$ for the genotypes, higher χ-ary alphabets could also be used. Then, a χ-ary alphabet is used for the string of length l instead of a binary alphabet. Therefore, instead of encoding 2^l different individuals with a binary alphabet, we are able to encode χ^l different possibilities. However, Goldberg (1990b) has shown that schema processing is maximum with binary alphabets.

Focusing on binary representations, we have a large number of different genotype-phenotype mappings that we can use as representations. If we use a redundancy-free encoding and want to encode 2^l phenotypes with 2^l possible genotypes, then there are $(2^l)!$ different possibilities for the genotype-phenotype mapping f_g (see Sect. 3.3.7). Nevertheless, for our comparison we want to focus on the three most widely used representations defined on binary strings:

- binary representation,
- Gray representation, and
- unary representation.

In contrast to the unary encoding, the binary and Gray encoding allows us to encode information redundancy-free. For the encoding of s possibilities, both encodings use $\log_2(s)$ bits (compare Sect. 3.1.1). The unary encoding uses $s - 1$ bits for encoding only s different possibilities and is a redundant representation. In the following paragraphs, we want to briefly review the important properties of the three different encodings:

When using the *binary encoding*, each integer phenotype $x^p \in \Phi_p = \{1, \ldots, x_{max}\}$ is represented by a binary genotype x^g of length $l = \lceil \log_2(x_{max}) \rceil$. The genotype-phenotype mapping f_g is defined as

$$x^p = f_g(x^g) = \sum_{i=0}^{l-1} 2^i x_i^g,$$

with x_i^g denoting the ith bit of x^g. Using the binary encoding for the integer one-max problem (5.1) results in the BinInt problem (compare Sect. 3.2.4).

Since the bits in the string are exponentially scaled, we must use the domino convergence model and GAs are affected by genetic drift (see Sect. 3.2). The bits are solved sequentially, and the low salient bits can be fixed randomly before they are reached by the solving process. Furthermore, the encoding has problems associated with the Hamming cliff (Schaffer et al. 1989). The Hamming cliff describes the effect that some neighboring phenotypes (the phenotypes have a distance of one) are represented by completely different genotypes (the distance between the genotypes is much larger than one). Therefore, the locality of the binary encoding is low. As a result, especially mutation-based search approaches have problems when using this encoding because they rely on a high locality of the encoding. We have seen in Sect. 3.3.5 that high locality is a necessary condition for a representation to preserve BB-complexity. Therefore, the ability of the binary encoding to preserve problem complexity is reduced in comparison to high-locality representations. However, the encoding also has some very interesting properties: It is linear, very compact and redundancy-free. For an example of the binary encoding, the reader is referred to Table 5.1

To overcome problems with the Hamming cliff and the different scaling of the alleles in binary strings, the *Gray encoding* was developed (Caruana and Schaffer 1988; Schaffer et al. 1989). When using Gray encoding, the average contribution of the genotypic alleles to the construction of the phenotype is the same for each allele in the binary genotype. Therefore, the Gray encoding is uniformly scaled (compare Sect. 3.2.1).

The Gray-encoded string itself can be constructed in two steps. At first, the phenotype is encoded using the binary encoding, and subsequently the binary-encoded string can be converted into the corresponding Gray-encoded genotype. The binary string $x^{bin} \in \{0,1\}^l = \{x_1^{bin}, x_2^{bin}, \ldots, x_l^{bin}\}$ is converted to the corresponding Gray-encoded string $x^{Gray} \in \{0,1\}^l = \{x_1^{Gray}, \ldots, x_l^{Gray}\}$ by the mapping $\gamma : \mathbb{B}^l \to \mathbb{B}^l$:

$$x_i^{Gray} = \begin{cases} x_i^{bin} & \text{if } i = 1, \\ x_{i-1}^{bin} \oplus x_i^{bin} & \text{otherwise,} \end{cases}$$

where $\oplus$ denotes addition modulo 2. The decoding of a Gray-encoded string is as follows:

$$x_i^{bin} = \bigoplus_{j=1}^{i} y_j^{Gray},$$

for $i = \{1, \ldots l\}$. As mentioned before, a Gray-encoded string has the same length as a binary-encoded string and the encoding is redundancy-free. Furthermore, the representation overcomes the problems with the Hamming cliff. All neighboring phenotypes are also neighboring genotypes. However, as the number of neighbors is different for the genotypic and phenotypic search space,

not all genotypic neighbors can correspond to phenotypic neighbors. Each phenotype has two neighbors (except $x^p = 0$ and $\max(x^p)$), whereas each genotype has $l = \lceil \log_2(\max(x^p)) \rceil$ neighbors. Therefore, there are more genotypic than phenotypic neighbors and the locality of the encoding is low ($d_m \neq 0$). However, when using the Gray encoding, for all genotypes $x^g \in \Phi_g$there exists a neighboring genotype $y^g \in \Phi_g$ who corresponds to the neighboring phenotype, $\{y^g | (d_{y^g,y^g} = 1) \wedge (d_{y^p,x^p} = 1)\}$. This property gives Gray encoding an advantage over binary encoding when using mutation-based operators like the bit-flipping operator (Whitley 1999) as there is always one genotypic mutation that results in a neighboring phenotype. In contrast, for the binary encoding there exist genotypes (for example $x^g = 1000$ which corresponds to $x^p = 8$) that do not have a genotypic neighbor which corresponds to the phenotypic neighbor (for example $x^g = 0111$ which corresponds to $x^p = 7$). For the binary encoding, mutation can not directly move from x^g to y^g although $d_{x^p,y^p} = 1$. For Gray encoding, there is always one direct move from x^g to y^g if $d_{x^p,y^p} = 1$. As before, Table 5.1 shows an example for the Gray encoding.

Finally, the *unary encoding* can be used for encoding integers. A phenotype x^p is encoded by the number u of ones in the corresponding genotype x^g. With the length $l = \max(x^p)$ of the string and x_i^g as the ith bit of x^g we get

$$x^p = f_g(x^g) = \sum_{i=0}^{l-1} x_i^g.$$

In contrast to the binary and Gray encoding, a string of length $l = s - 1$ is necessary for representing s different phenotypes x^p. Therefore, the genotype-phenotype mapping is no longer a one-to-one mapping but redundant. When encoding the phenotypic space $\Phi_p = \{0, 1, \ldots, l\}$ using a unary string of length l, each of the $l + 1$ phenotypes $x^p \in \Phi_p$ is represented by $\binom{l}{x^p}$ different genotypes x^g. The number of genotypes that represent x^p is illustrated for $l = 7$ in Fig. 5.2. Some phenotypes are represented by only one genotype ($x^p = 0$ and $x^p = 7$), whereas $x^p = 3$ and $x^p = 4$ are represented by 35 genotypes.

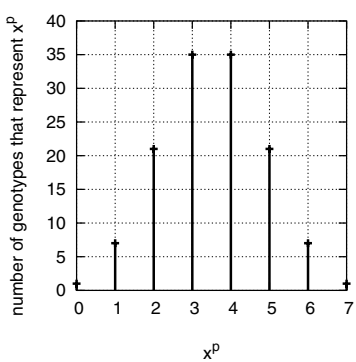


Figure 5.2. Redundancy of the unary encoding for $l = 7$

The unary encoding is uniformly scaled and phenotypic neighbors correspond to genotypic neighbors. However, it has low locality as there are more genotypic than phenotypic neighbors (compare the discussion of the Gray encoding from above), and some genotypic neighbors do not correspond to phenotypic neighbors. We can illustrate this with an example. $x^p = 2$ and $y^p = 3$ are neighboring phenotypes. The corresponding genotypes $x^g = 0011$ and $y^g = 0111$ are also neighbors. However, other corresponding genotypes such as $x^g = 0011$ and $y^g = 1110$ are not neighbors. As the locality is low, the encoding also has low distance distortion. As not all genotypes that correspond to the same phenotype are similar to each other, the unary encoding is a non-synonymously redundant encoding (compare Sect. 3.1.2).

Finally, we want to give a brief example for the three different types of encodings. Table 5.1 illustrates how the phenotypes $x^p \in \{0, \ldots, 7\}$ can be represented by the binary, Gray, and unary encoding.

x^p	x^g		
	binary	Gray	unary
0	000	000	0000000
1	001	001	0000001, 0000010, ...,0100000, 1000000
2	010	011	0000011, 0000101, ...,1010000, 1100000
3	011	010	0000111, 0001011, ...,1101000, 1110000
4	100	110	0001111, 0010111, ...,1110100, 1111000
5	101	111	0011111, 0101111, ...,1111010, 1111100
6	110	101	0111111, 1011111, ...,1111101, 1111110
7	111	100	1111111

Table 5.1. An example for using binary, Gray, and unary encodings

5.3 A Theoretical Comparison

This section uses the framework of representations we presented in Chap. 4 to theoretically compare the performance of GEAs using binary, Gray and unary representations. The framework allows us to make predictions about the performance of GEAs which will be empirically verified in Sect. 5.4. In particular, we illustrate the effects of the non-uniform redundancy of the unary encoding, how the genetic search process is prolonged by the effect of exponentially scaled BBs for the binary encoding, and how the complexity of the problem is not well preserved by the Gray and binary encoding.

5.3.1 Redundancy and the Unary Encoding

We know from Sect. 3.1 that redundancy reduces GEA performance if the encoding underrepresents good solutions. Furthermore, we know from the previous section (see Fig. 5.2) that the unary representation is a non-synonymously

redundant encoding as one phenotype is encoded on average by more than one genotype but not all genotypes that encode one phenotype are similar to each other. Due to its non-synonymous redundancy, the use of the unary encoding randomizes genetic search and the genotypic problem difficulty is different from the phenotypic problem difficulty. In general, when using non-synonymous redundant encodings, phenotypically easy problems become more difficult whereas phenotypically difficult problems become more easy.

However, when using the unary encoding for the integer one-max and deceptive trap problem, the integer one-max problem remains genotypically fully easy for recombination-based search (Goldberg 1989b; Deb and Goldberg 1993; Deb and Goldberg 1994) as all schemata containing the global optimum are still superior to their competitors (compare also Sects. 3.3.6 and 3.3.7). It also remains easy for mutation as it is a unimodal problem for mutation-based search and the structure of the fitness landscape guides mutation towards the global optimum. Analogously, the fully deceptive trap remains fully difficult. The only real handicap of the unary encoding seems to be the non-uniform redundancy. Therefore, we want to neglect the non-synonymous redundancy and focus on the overrepresentation and underrepresentation of the optimal solution.

We want to predict the performance of GEAs using the unary encoding for the integer one-max and integer deceptive trap problem from Sect. 5.1. We assume that $|\Phi_p| = s$. Therefore, for both problems, the integer one-max and the integer deceptive trap problem, the length of the unary encoded string is $l = s - 1$. Thus, 2^{s-1} different genotypes only encode s different phenotypes. $\log_2(s)$ Bits of information content (see Sect. 3.1.2) are encoded by $s - 1$ bits. Therefore, we get for the order of redundancy (see Sect. 3.1.2)

$$k_r = \frac{s-1}{\log_2(s)} \quad \text{, for } s > 1.$$

On average, k_r bits of a unary encoded bitstring are necessary for encoding one Bit of information content. This means, on average 2^{k_r} different genotypes represent only two different phenotypes.

When using the unary encoding for the integer one-max or deceptive trap defined in Sect. 5.1, the optimal phenotype ($x^{p,opt} = l$) is represented by only one genotype (a string of only ones). Therefore, the number of genotypic BBs that represent the best phenotypic BB is $r = 1$.

From (3.9), we get for the population size $N = O\left(\frac{2^{k_r}}{r}\right)$ when using redundant encodings. Therefore, the necessary population size N when using unary encoding for the integer one-max and deceptive trap problem is increased in comparison to an encoding with uniform or no redundancy as

$$N_{unary} = N_{redundancyfree} \times 2^{\frac{s-1}{\log_2(s)}-1}.$$

The equation shows that with increasing string length $l = s-1$, the necessary population size when using unary encoding increases exponentially. This effect

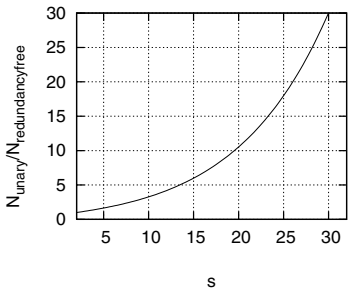


Figure 5.3. Necessary population size N_{unary} when using unary encoding

is illustrated in Fig. 5.3. For even small problems the necessary population size N_{unary} is unreasonably high. Obviously the use of the unary encoding results for the proposed integer one-max and deceptive trap problem in a low GA performance.

However, we know from Sect. 3.1 that the performance of non-uniformly redundant encodings depends on the specific problem they are used for. GEAs using synonymously redundant encodings only show low performance if the good solutions are underrepresented. Therefore, we want to investigate in the remaining paragraphs for which problems the unary encoding performs well.

For both problems, the integer one-max and deceptive trap problem, the optimal solution $x^{p,opt} = l$ is strongly underrepresented by only one genotype and GEAs using the unary encoding show low performance. When changing the position of the optimal solution $x^{p,opt}$ (like in the gen-one-max problem) and the optimal solution is not a string with only ones (or only zeros), GEAs can perform better when using the unary encoding. This means, if we use a different integer fitness function f_p and the optimal solution would be, for example, $x^{p,opt} = l/2$ (resulting in a gen-one-max problem with $a = l/2$), then the optimal solution would be strongly overrepresented and GEAs using the unary encoding would be able to solve this problem very effectively.

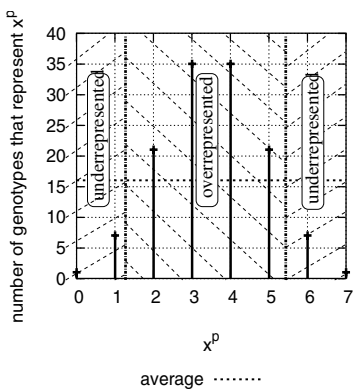


Figure 5.4. Areas of over- and underrepresented phenotypes when using unary encoding

We want to illustrate the problem of the unary encoding with underrepresented solutions in Fig. 5.4 more clearly. On average

$$\gamma_{avg} = \frac{1}{s} \sum_{x^p=0}^{s-1} \binom{s-1}{x^p}$$

genotypes represent one phenotype. Therefore, if $\binom{s-1}{x^{p,opt}} < \gamma_{avg}$ the optimal solution $x^{p,opt}$ is underrepresented and the performance of the GA is reduced. However, if $\binom{s-1}{x^{p,opt}} > \gamma_{avg}$ the optimal solution is overrepresented and redundancy helps the GA in finding the optimal solution $x^{p,opt}$.

We see that the performance of GEAs using the unary encoding depends on the structure of the optimization problem we want to solve. The unary encoding can be a good choice for the integer one-max problem or the integer deceptive trap problem if the optimal solution is not strongly underrepresented.

5.3.2 Scaling, Modification of Problem Difficulty, and the Binary Encoding

In Sect. 3.2, we illustrated how the search process of GEAs is prolonged by using non-uniformly scaled encodings. If integers are encoded using the binary representation, domino convergence occurs and GEAs are affected by genetic drift. As a result, the probability α of GEA failure increases for small population sizes N. However, scaling only affects GEAs for the easy integer one-max problem because the optimal solution can be found even with small populations, but not for the more difficult deceptive trap problem. Here, the necessary population size is large enough that no drift occurs. For further details regarding the effects of exponentially scaled encodings, such as the binary encoding on GEA performance, we refer to Sect. 3.2 as well as to results presented in the literature (Thierens 1995; Thierens et al. 1998; Lobo et al. 2000).

The performance of GEAs using binary encoding is not only affected by the exponential scaling of the encoding, but also by problems associated with locality and the Hamming cliff (Caruana and Schaffer 1988; Caruana et al. 1989; Schaffer et al. 1989). Binary encodings have low locality ($d_m \neq 0$) as not all neighboring genotypes correspond to neighboring phenotypes. As an example we can chose the genotypes $x^g = 011$ and $x^g = 111$, which are neighbors and have distance $d_{x^g,y^g} = 1$. However, the corresponding phenotypes $x^p = 3$ and $y^p = 7$ have distance $d_{x^p,y^p} = 4$. As we know from Sect. 3.3.5 that low-locality encoding change problem difficulty, we expect that the non-redundant binary encoding changes the structure and complexity of the BBs. How exactly the structure of the BBs is changed is exemplarily measured in the following paragraphs (compare Table 5.3).

5.3.3 Modification of Problem Difficulty and the Gray Encoding

The non-redundant Gray encoding has low locality ($d_m \neq 0$), as the genotypes have a larger number of neighbors than the phenotypes. Therefore, the complexity of BBs is modified and the problem difficulty for GEAs is changed (compare Sect. 3.3) when mapping phenotypic integers on genotypic bitstrings. As a result, fully easy integer problems remain not fully easy.

Mutation-based search approaches using Gray encoding perform better than using the binary encoding, as there is always one genotypic mutation that allows the search method to reach a neighboring phenotype in one search step. This performance advantage of Gray encoding in comparison to binary encoding has already been described in other work (Whitley et al. 1997; Rana and Whitley 1997; Whitley and Rana 1997; Rana and Whitley 1998; Whitley 1999; Whitley 2000a; Whitley 2000b). This work formulated a Free-Lunch theorem for the use of Gray encoding and mutation-based search approaches. GEAs using mutation as the main search operator perform better on easy problems (these are the problems which we are interested in) when using Gray encoding than when using binary encoding. It was shown that the number of local optima introduced by Gray encoding is smaller than by binary encoding.

We know that using Gray encoding changes the problem difficulty for GEAs. When using crossover-based search, the schema analysis is an appropriate method to measure problem difficulty (compare Sect. 2.3.2). For both, binary and Gray encoding, $d_c \neq 0$ and genotypic and phenotypic problem difficulty is different when using crossover-based search approaches. The distance distortion d_c is high, as the structure of the genotypic and phenotypic search space is different and a phenotype has a lower number of neighbors than a genotype. The following analysis of the schemata fitness reveals for the integer one-max and deceptive trap problem that in comparison to the binary encoding, Gray encoding does not preserve the complexity of BBs as well. This leads to a lower GA performance.

To investigate how well BB-complexity is preserved, we analyze the fitness of the schemata for a 3-bit problem ($s = 2^3 = 8$) using Gray versus binary encoding. In Table 5.2, we present the binary and Gray-encoded genotypes, and the resulting fitness values for the integer one-max and deceptive trap problem. In Table 5.3, we present the average fitness of the schemata for the two problems. Reviewing problem complexity, the problem is fully deceptive if all schemata of lower order containing the global optimum are inferior to their competitors (Deb and Goldberg 1994). Analogously, the problem is fully easy if all schemata containing the global optimum are superior to their competitors (compare Sect. 2.3.2).

The analysis shows that for the fully easy integer one-max problem with binary encoding, all schemata containing the global optimum $x^{g,opt} = 111$ are superior to their competitors. Although the binary encoding has low locality, the fully easy integer one-max problem remains fully easy, and the binary encoding preserves the difficulty of the problem well. The schema analysis for

Table 5.2. Using binary and Gray encoding for an integer one-max and deceptive trap problem ($s = 8$). The resulting length of the genotypes $l = 3$.

genotype x^g	binary	000	001	010	011	100	101	110	111
	Gray	000	001	011	010	110	111	101	100
phenotype x^p	integer	0	1	2	3	4	5	6	7
fitness	$f_{one-max}(x^p)$	0	1	2	3	4	5	6	7
	$f_{deceptive}(x^p)$	6	5	4	3	2	1	0	7

Table 5.3. Schemata fitness for the integer one-max and deceptive trap problem using binary versus Gray encoding. The integer one-max problem remains fully easy when using the binary representation. Using Gray encoding makes the problem more difficult as some of the high quality schemata have the same fitness as the misleading schemata. The situation for the deceptive trap is the opposite one. The fully difficult deceptive trap becomes easier to solve when using Gray encoding.

		order	3	2			1			0
integer one-max problem ($s = 8$)	binary	schema	111	11*	1*1	*11	**1	*1*	1**	***
		fitness	**7**	**6.5**	**6**	**5**	**11**	**4.5**	**5.5**	**3.5**
		schema		01*	0*1	*01	**0	*0*	0**	
		fitness		2.5	2	3	3	2.5	1.5	
		schema		10*	1*0	*10				
		fitness		4.5	5	4				
		schema		00*	0*0	*00				
		fitness		0.5	1	2				
	Gray	schema	100	10*	1*0	*00	1**	*0*	**0	***
		fitness	**7**	**6.5**	**5.5**	**3.5**	**5.5**	**3.5**	**3.5**	**3.5**
		schema		11*	1*1	*11	0**	*1*	**1	
		fitness		4.5	**5.5**	**3.5**	1.5	**3.5**	**3.5**	
		schema		01*	0*1	*01				
		fitness		2.5	1.5	**3.5**				
		schema		00*	0*0	*00				
		fitness		0.5	1.5	**3.5**				
integer deceptive trap problem ($s = 8$)	binary	schema	111	11*	1*1	*11	**1	*1*	1**	***
		fitness	**7**	3.5	4	**5**	**4**	2.5	2.5	**3.5**
		schema		01*	0*1	*01	**0	*0*	0**	
		fitness		3.5	4	3	3	**3.5**	**4.5**	
		schema		10*	1*0	*10				
		fitness		1.5	1	2				
		schema		00*	0*0	*00				
		fitness		**5.5**	**5**	4				
	Gray	schema	100	10*	1*0	*00	1**	*0*	**0	***
		fitness	**7**	3.5	**4.5**	**6.5**	2.5	**4.5**	**4.5**	**3.5**
		schema		11*	1*1	*11	0**	*1*	**1	
		fitness		1.5	0.5	2.5	**4.5**	2.5	2.5	
		schema		01*	0*1	*01				
		fitness		3.5	**4.5**	2.5				
		schema		00*	0*0	*00				
		fitness		**5.5**	**4.5**	2.5				

the Gray encoding reveals that the schemata containing the global optimum $x^{g,opt} = 100$ are not always superior to their competitors. Therefore, the problem is not fully easy anymore, and the Gray encoding changes problem difficulty and does not preserve the easiness of the integer one-max problem.

The schemata analysis of the integer trap problem reveals that the problem remains not fully deceptive when using the binary encoding. Some of the schemata containing the global optimum $x^{g,opt} = 111$ are superior to their competitors (*11 and **1). However, when using Gray encoding even more schemata containing the global optimum are not inferior to their competitors (1*0, *00, *0*, **0). The phenotypically fully difficult problem is not fully difficult anymore.

5.4 Experimental Results

In this section, we present an experimental verification of the performance differences between the three different representations we discussed in the previous section.

5.4.1 Integer One-Max Problem and Deceptive Integer One-Max Problem

We compare the performance of GAs using binary, Gray, and unary encoding for the integer one-max and deceptive trap problems as defined in Sect. 5.1. We performed 250 runs for each problem instance and each run was stopped after the population was fully converged (all individuals in the population are the same). For the integer one-max problem we used uniform crossover, and for the integer deceptive trap we used two-point crossover. As selection method we used tournament selection without replacement of size two. We used no mutation as we want to focus on selectorecombinative GEAs.

The Figs. 5.5, 5.6, 5.7, and 5.8 present results for the integer one-max problem, and the Figs. 5.9 and 5.10 for the integer deceptive trap problem. The plots show for different representations the proportion of correctly solved sub-problems at the end of the run (left) and the run duration t_{conv} (right) with respect to the population size N. For the one-max problem, we concatenated 20 sub-problems of order 2 ($s = 2^2 = 4$, see Fig. 5.5), 3 ($s = 8$, Fig. 5.6), 4 ($s = 16$, Fig. 5.7), and 5 ($s = 32$, Fig. 5.8)[1]. The fitness of an individual is calculated as the sum of the fitness of the 20 concatenated sub-problems. Because large integer deceptive traps are not solvable by GAs in a reasonable time, we only present results for the deceptive trap problem of order 2 ($s = 4$, Fig. 5.9), and 3 ($s = 8$, Fig. 5.10). Using binary or Gray encoding results for the order 2 problems in a string length $l = 40$, for order 3 in $l = 60$, for order

[1]The order r of a problem is defined as $r = \log_2 s$ and describes the length of the corresponding binary- or Gray-encoded string.

4 in $l = 80$, and for order 5 in $l = 100$. When using unary encoding we need $20 \times 3 = 60$ bits for order 2, $20 \times 7 = 140$ bits for order 3, $20 \times 15 = 300$ bits for order 4, and $20 \times 31 = 620$ bits for order 5 problems.

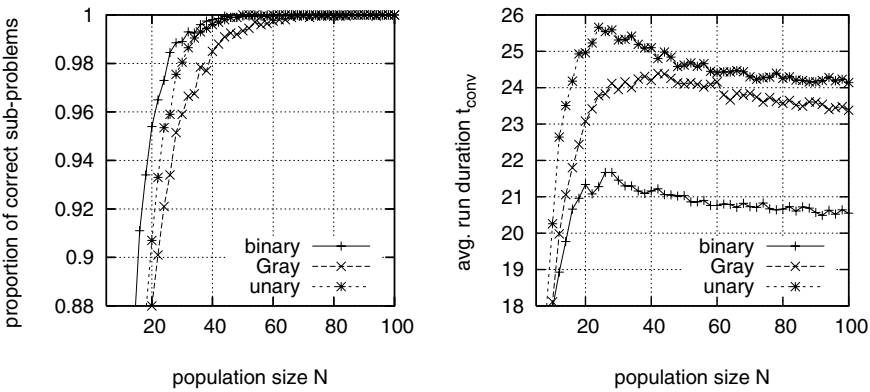


Figure 5.5. Integer one-max problem of order 2. We concatenated $m = 20$ sub-problems and the size of the search space $|\Phi_p| = 2^2 = 4$. We show the average proportion of correct sub-problems at the end of run (left) and the average running time (right). Due to the low complexity of the problems all three representations perform about the same. However, binary encoding is much faster in finding the good solutions.

Due to the problems of the unary encoding with redundancy, which result in an underrepresentation of the optimal solution, GAs using unary encoding show decreasing performance with increasing problem size. Therefore, for one-max problems of order more than three the GA performance is significantly worse than when using Gray or binary encoding. Although the one-max problem remains fully easy, GEA performance is reduced because the optimal solution is strongly underrepresented. Only for the almost trivial one-max problem of order 2 or 3 has the unary encoding a comparable performance. The plots nicely illustrate that only for small one-max problems the benefits from the preservation of BB-complexity can compensate the performance reduction caused by the underrepresentation of the optimal solution. For deceptive traps of order more than 2, unary encoding fails completely because the problem remains fully difficult and the optimal solution is underrepresented. Furthermore, the plots show that due to the preservation of BB-complexity, a GA using unary encoding performs in comparison to Gray or binary encoding relatively better for the easy one-max problem than for the deceptive trap. The failure of the encoding for the deceptive trap can be better understood if we recognize that an order 3 problem results in a fully deceptive BB of length $l = 7$. This problem is only solvable with much larger population sizes.

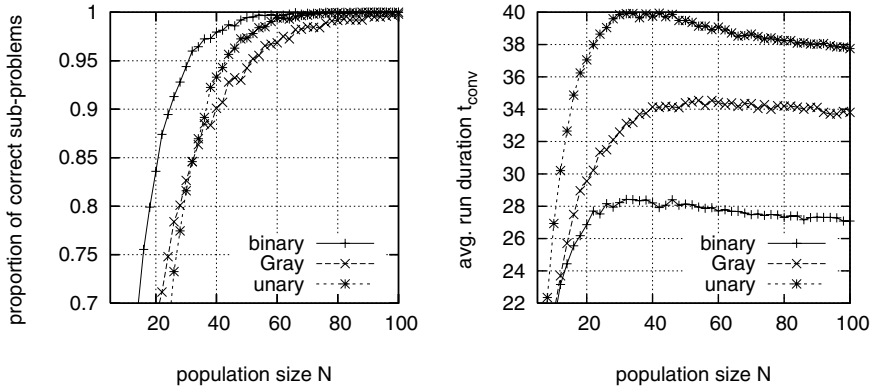


Figure 5.6. Integer one-max problem of order 3. We concatenated $m = 20$ sub-problems and the size of the search space $|\Phi_p| = 2^3 = 8$. We show the average proportion of correct sub-problems at the end of run (left) and the average running time (right). Binary encoding performs the best. Because the optimal solutions are underrepresented, GAs using the unary encodings perform worse than when using Gray encoding for small population sizes.

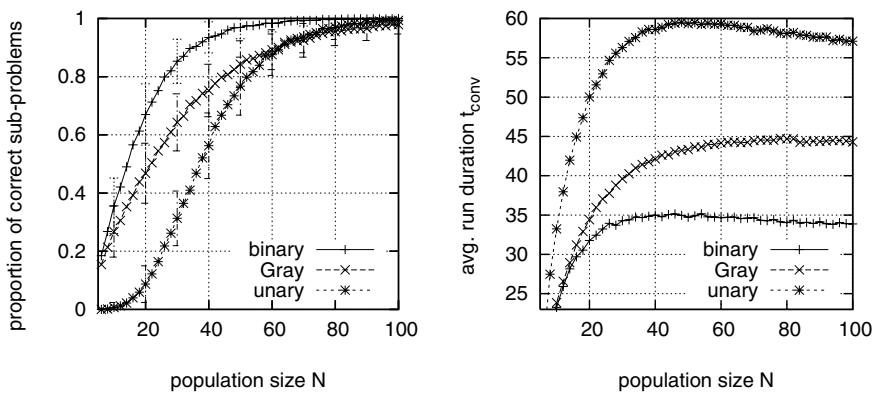


Figure 5.7. Integer one-max problem of order 4. We concatenated $m = 20$ sub-problems and the size of the search space $|\Phi_p| = 2^4 = 16$. We show the average proportion of correct sub-problems at the end of run (left) and the average running time (right). Because the binary encoding preserves BB-complexity better than Gray encoding, a GA using binary representations performs best. Because of problems with redundancy, the unary encoding performs worst and needs the most fitness evaluations. The error bars indicate the standard deviation of some results.

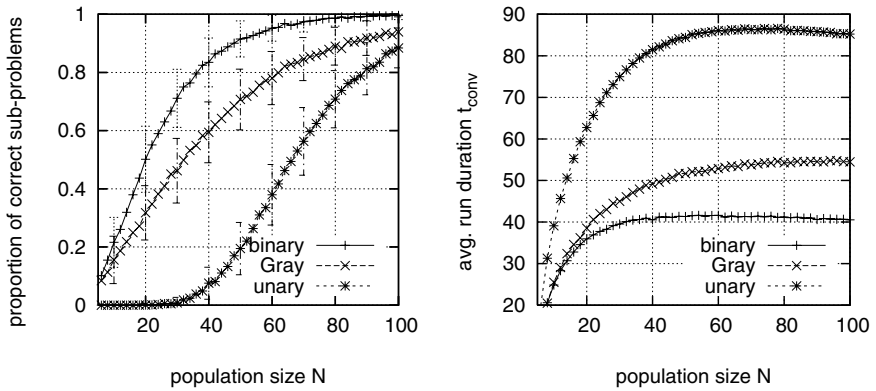


Figure 5.8. Integer one-max problem of order 5. We concatenated $m = 20$ sub-problems and the size of the search space $|\Phi_p| = 2^5 = 32$. We show the average proportion of correct sub-problems at the end of run (left) and the average running time (right). As before, binary encoding performs best. It becomes obvious that with increasing problem size GEAs using unary encoding have increasing difficulty in finding the good solutions. Furthermore, the performance differences between the binary and Gray encoding become larger with increasing problem size. The error bars indicate the standard deviation of some results.

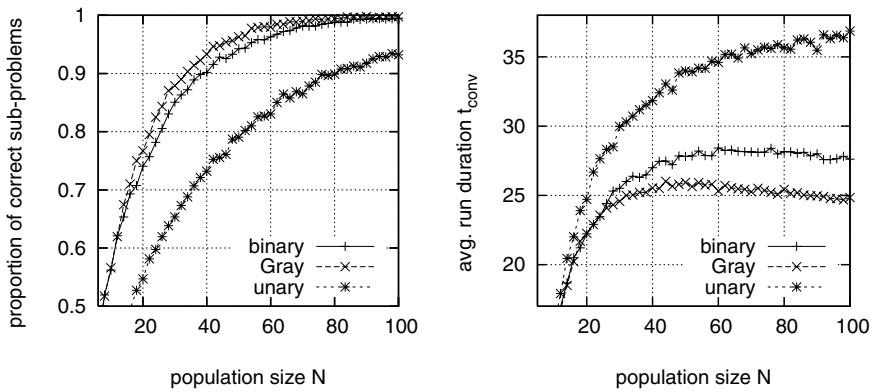


Figure 5.9. Integer deceptive trap problem of order 2. We concatenated $m = 20$ sub-problems and the size of the search space $|\Phi_p| = 2^2 = 4$. We show the average proportion of correct sub-problems at the end of run (left) and the average running time (right). Gray encoding performs slightly better than binary encoding as it preserves the structure of the sub-problems worse. Because unary encoding strongly underrepresents the optimal solution, it performs the worst.

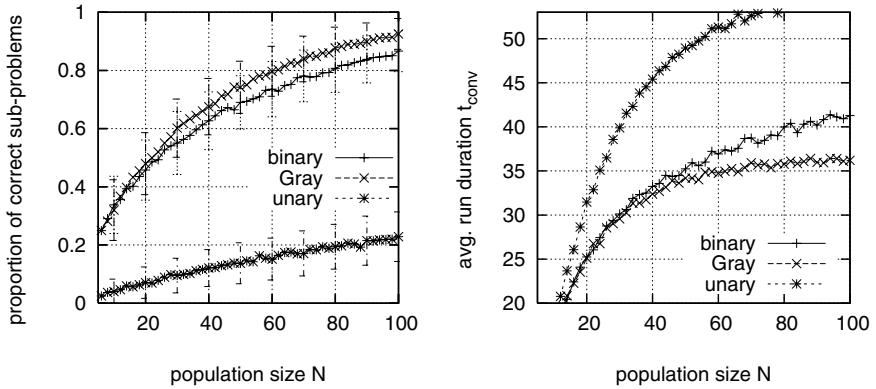


Figure 5.10. Integer deceptive trap problem of order 3. We concatenated $m = 20$ sub-problems and the size of the search space $|\Phi_p| = 2^3 = 8$. We show the average proportion of correct sub-problems at the end of run (left) and the average running time (right). Gray encoding performs significantly better than binary encoding as it makes the fully difficult deceptive trap problem easier to solve. Unary encoding fails as it has problems with redundancy. The error bars indicate the standard deviation of some results.

As expected, Gray encoding performs worse than binary encoding for the one-max problem, and better for the deceptive trap problem. Because for the integer one-max and deceptive trap problem Gray encoding preserves BB-complexity less than binary encoding (compare Table 5.3), the fully easy integer one-max problem becomes more difficult to solve, whereas the fully difficult deceptive trap is easier to solve for a GA using Gray encoding.

Finally, the influence of exponentially-scaled representations like the integer encoding on the performance of GEAs can be seen for the one-max problem. For small population sizes N, genetic drift has a larger impact on the solution quality. Therefore, for the easy integer one-max problem and small population sizes N, GAs using binary encoding perform only slightly better than Gray encoding. For larger population sizes, however, the effect of genetic drift is reduced and GAs using binary representation perform relatively better.

We see that the empirical results nicely verify the theoretical predictions from the previous section. Fig. 5.11 summarizes some of the results for the integer one-max problem and shows the proportion of correct sub-problems at the end of the run over the order of the problem. Due to the underrepresentation of the optimal solutions the performance of a GA using unary encoding strongly decreases with increasing problem size. A GA using binary encoding performs best in comparison to Gray and unary encoding, as the exponential scaling of the representation affects a GA only for small populations, and the encoding preserves BB-complexity better than the Gray encoding. As a result,

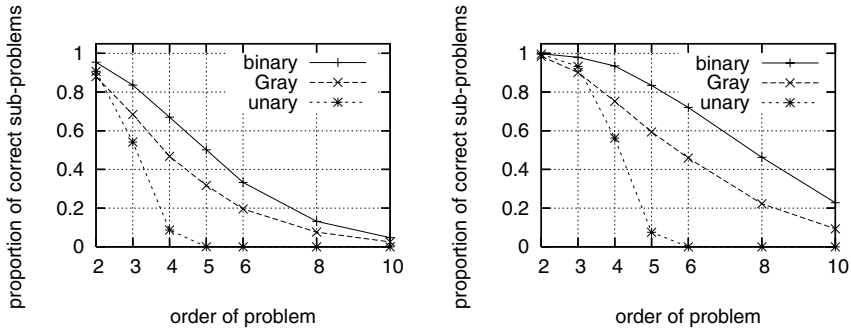


Figure 5.11. Proportion of correct sub-problems at the end of the run over the order of the problem for a population size of 20 (left) and 40 (right) for binary, Gray, and unary encoding. The figures are plotted for the integer one-max problem. It can be seen that with increasing order of the problem the performance of the unary representation strongly decreases. When using Gray or binary encoding the performance of the GA declines much less.

the easy one-max problem remains easier with the binary encoding than with the Gray encoding. Although Gray encoding preserves BB-complexity worst, it still significantly outperforms unary encoding which fails for the one-max and deceptive trap due to the underrepresentation of the optimal solution.

5.4.2 Modifications of the Integer One-Max Problem

This section investigates how the performance of Gray and binary encoding depends on the properties of the optimal solution for mutation-based and crossover-based search. For the experiments we use the gen-one-max problem as defined in (5.2).

Mutation-Based Search Using Simulated Annealing

We investigate how the locality of an encoding influences the performance of mutation-based search approaches. In our investigations, we assume that the gen-one-max problem defined in (5.2) is easy for mutation-based search independently of the position of the optimal solution a.

We want to use simulated annealing (SA) as a representative for a mutation-based search approach because it uses only mutation, and can in contrast to for example an (1+1) evolution strategy, solve difficult, and multimodal, problems more easily. Simulated annealing can be modeled as a GA with population size $N = 1$ and Boltzmann selection (Mahfoud and Goldberg 1995). In each generation, a genotypic offspring $x^{g,o}$ is created by applying mutation to the parent $x^{g,p}$. Therefore, if we use bit-flipping-mutation, $x^{g,o}$ always has genotypic distance 1 to its parent $x^{g,p}$. If $x^{g,o}$ has higher fitness

than $x^{g,p}$, it replaces $x^{g,p}$. If it has lower fitness, it replaces $x^{g,p}$ with probability $P(T) = \exp\left(-\frac{f(x^{g,o})-f(x^{g,p})}{T}\right)$. By lowering the temperature T, the probability of accepting worse solutions decreases. For further information the reader is referred to van Laarhoven and Aarts (1988).

For our investigation we concatenate 10 integer gen-one-max problems of different length l. When using Gray or binary encoding, each of the 10 phenotypic integers $x^p \in \{0, \ldots, 2^l - 1\}$ corresponds to l bits in the genotype. Therefore, for $l = 5$, the overall length of a genotype is 50. The fitness of an individual is calculated as the sum over the fitness of the 10 sub-problems. The fitness of one sub-problem is calculated according to (5.2).

Figure 5.12 presents results using SA for two instances ($a = 15$ and $a = 31$) of the gen-one-max problem with $l = 5$. We show the number of correctly solved sub-problems over the number of fitness evaluations. The start temperature $T_{start} = 50$ is reduced in every step by the factor 0.995. Therefore, $T_{t+1} = 0.995 \times T_t$. Mutation is defined to randomly change one bit in the genotype. We performed 100 runs and each run was stopped after 2000 mutation steps. The results show that mutation-based search approaches using Gray encoding always solve all 10 sub-problems. In contrast, for $a = 15$, mutation-based search using binary encoding gets stuck in local optima because the optimal solution lies in areas with problems with the Hamming cliff.

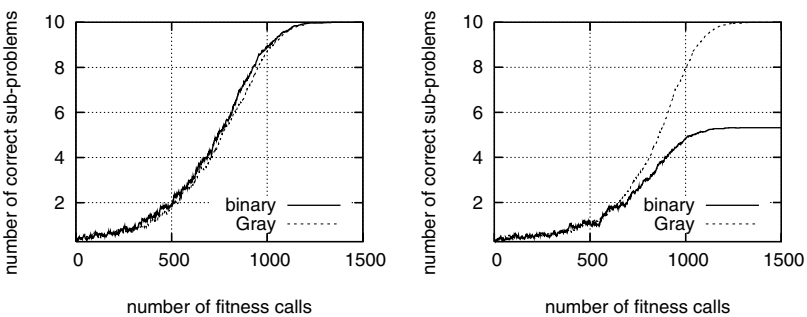


Figure 5.12. We use SA and show the number of correctly solved sub-problems over the number of fitness calls (search steps) for $a = 31$ (left) and $a = 15$ (right).

To generalize our investigation and to determine how the performance of mutation-based search depends on the structure of the integer optimization problem, Fig. 5.13 illustrates how SA performance depends on the value of the optimal solution a. We show results for $l = 3$ (left) and $l = 5$ (right). We only change a and use the same parameter settings as before. The plots show that using Gray encoding allows SA to reliably find the optimal solution independently of the location of the best solution. Using binary encoding often results in lower SA performance and the SA gets stuck in local optima.

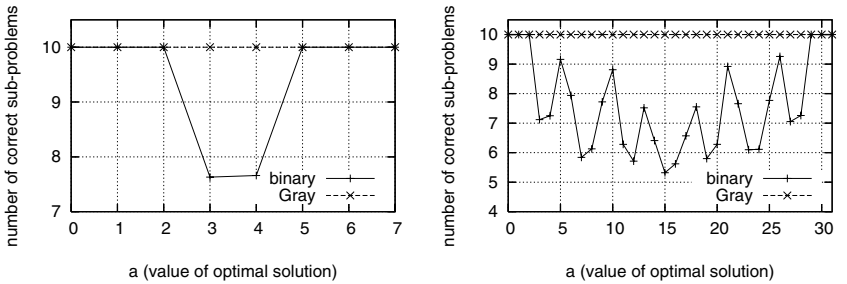


Figure 5.13. We use SA and show the number of correctly solved sub-problems at the end of a run over the location of the optimal solution a for $l = 3$ (left) and $l = 5$ (right).

This performance differences between Gray and binary encoding has already been observed in other work (Schaffer et al. 1989; Whitley et al. 1997; Whitley 1999). The binary encoding has problems associated with the Hamming cliff and has low locality. Due to its low locality, small changes of the genotype do not always result in small changes of the corresponding phenotype. In addition, due to the Hamming cliff, neighboring phenotypes are not always neighboring genotypes (for example $x^p = 15$ and $x^p = 16$). Therefore, mutation-based search can never move directly from $x^p = 15$ to $x^p = 16$. Only if the optimal solution consists of almost only ones or zeros does binary encoding show similar performance to Gray encoding.

We can explain the performance differences between Gray and binary encoding by focusing on the Hamming distances between neighboring individuals. Although both encodings, Gray and binary encoding, have low locality, and not all genotypic neighbors correspond to phenotypic neighbors, there are some differences in the neighborhood structure. Table 5.4 presents the properties of neighboring individuals for Gray versus binary encoding for $l = 3$. We show the $l = 3^2 = 8$ possible phenotypes x^p and their phenotypic neighbors y^p, $\{y^p | d_{x^p,y^p} = 1\}$. Furthermore, we show for Gray and binary encoding the corresponding genotypes x^g, the genotypes y^g that correspond to the phenotypic neighbors y^p, and the average genotypic Hamming distances $avg(d^g_{x^g,y^g})$ between x^g and y^g, where

$$avg(d^g_{x^g,y^g}) = \frac{1}{|n^p|} \sum_{\{y^g | d_{x^p,y^p}=1\}} d_{x^g,y^g}.$$

$|n^p|$ denotes the number of phenotypic neighbors y^p.

As already discussed in Sect. 5.2, for Gray-encoded genotypes there is always a neighboring genotype that corresponds to a phenotypic neighbor. Therefore, only one genotypic mutation step is necessary to reach all possible phenotypic neighbors and the average distances between x^g and the genotypes y^g that represent the phenotypic neighbors y^p is one ($avg(d^g_{x^g,y^g}) = 1$).

Table 5.4. Properties of neighboring individuals for Gray and binary encoding

x^p		0	1	2	3	4	5	6	7
$\{y^p \mid d_{x^p,y^p} = 1\}$		1	0, 2	1, 3	2, 4	3, 5	4, 6	5, 7	6
Gray	x^g	000	001	011	010	110	111	101	100
	$\{y^g \mid d_{x^p,y^p} = 1\}$	001	000,011	001,010	011,110	010,111	110,101	111,100	101
	$avg(d^g_{x^g,y^g})$	1	1	1	1	1	1	1	1
binary	x^g	000	001	010	011	100	101	110	111
	$\{y^g \mid d_{x^p,y^p} = 1\}$	001	000,010	001,011	010,100	011,101	100,110	101,111	110
	$avg(d^g_{x^g,y^g})$	1	1.5	1.5	2	2	1.5	1.5	1

For binary encoding, $avg(d^g_{x^g,y^g}) \geq 1$. The average genotypic distance between the genotypes x^g and the genotypes y^g that represent the phenotypic neighbors of x^g increases for $x^p \to (x^p_{max}/2)$. For x^p_{max} and x^p_{min} it is minimal ($avg(d^g_{x^g,y^g}) = 1$), and for $x^p = x^p_{max}/2$ it is maximal ($avg(d^g_{x^g,y^g}) = (1+l)/2$, where $l = s - 1$ is the length of the genotype). As a result, the gen-one-max problem is, independently of the position of the optimal solution, easy for mutation-based search using Gray encoding as there is always a neighboring genotype which corresponds to a neighboring phenotype. In contrast, for the binary encoding, there is not always a neighboring genotype that corresponds to the phenotypic neighbors. As $avg(d^g_{x^g,y^g})$ increases for $a \to (x^p_{max}/2)$, the difficulty of gen-one-max problems for mutation-based search increases with $a \to (x^p_{max}/2)$. This behavior of mutation-based search using the binary encoding can be nicely observed in Fig. 5.13.

Crossover-Based Search Using Genetic Algorithms

We investigate for the gen-one-max problem how Gray and binary encoding influence the performance of crossover-based search.

We have seen discussed in Sect. 5.2 that both representations have low locality; however, Gray encoding better preserves the phenotypic neighborhood structure (compare results from the previous paragraphs). Due to the low locality of the encodings, offspring produced by standard crossover mechanisms could have nothing in common with their parents. For example, if we use binary encoding and uniform crossover we can get from the parents $x^p = 4$ ($x^g = 100$) and $y^p = 3$ ($y^g = 011$) the offspring $z^p = 7$ ($z^g = 111$). The offspring has phenotypically nothing in common with its parents and the phenotypic distances between the offspring and its parents are much larger than the distances between both parents (compare Sect. 3.3.5). Therefore, both encodings change the difficulty of the easy integer problem.

As before, we concatenate 10 integer gen-one-max problems with $l = 5$ ($x^p \in \{0, \ldots 31\}$). For our investigation we use a selectorecombinative standard GA (Goldberg 1989c) using only uniform crossover and no mutation. The population size is set to $N = 20$ and we use tournament selection without replacement of size 2. We performed 100 runs, and each run was stopped after the population was fully converged.

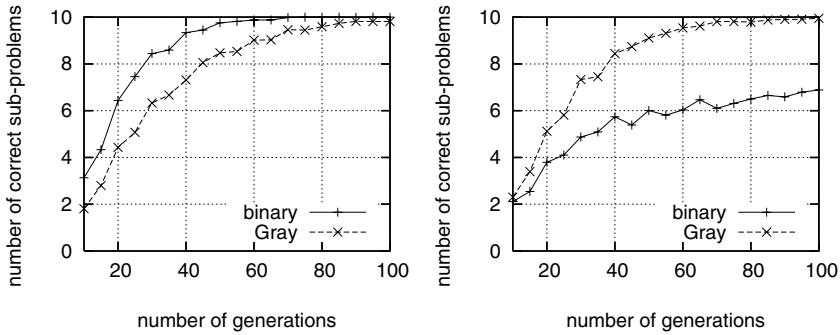


Figure 5.14. We use a selectorecombinative GA and show the number of correctly solved sub-problems over the number of generations for a=31 (left) and a=15 (right).

In Fig. 5.14, we show the number of correctly solved sub-problems over the number of generations for $a = 31$ (left) and $a = 15$ (right). For $a = 31$, the gen-one-max problem is equivalent to the one-max problem and we get the same results as presented in Sect. 5.4.1. The results show that selectorecombinative GAs using binary encoding outperform Gray encoding for $a = 31$. For $a = 15$, the situation changes and GAs using Gray encoding perform significantly better than binary encoding.

As before, we want to generalize our investigation and show in Fig. 5.15 how the average number of correctly solved sub-problems at the end of the run depends on the value of the optimal solution a. We show results for $l = 3$ (left) and $l = 5$ (right). It can be seen that GAs using binary encoding perform better than Gray encoding if a is either small ($a \to x^p_{min}$) or large ($a \to x^p_{max}$). Otherwise, GAs using Gray encoding perform better.

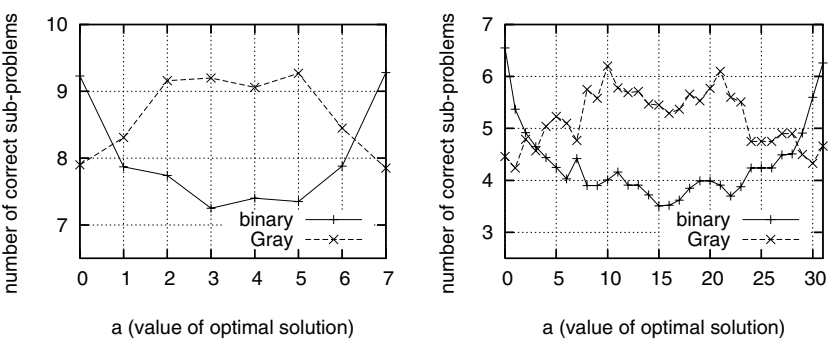


Figure 5.15. We use a selectorecombinative GA and show the number of correctly solved sub-problems at the end of a run over the location of the optimal solution a for $l = 3$ (left) and $l = 5$ (right).

When using crossover-based search, the difficulty of the original optimization problem f_p only remains unchanged if the genotype-phenotype mapping f_g does not change the distances between the individuals ($d_c = 0$). However, both encodings, Gray and binary, have high distance distortion (and low locality) and change the distances between corresponding genotypes and phenotypes. Therefore, both encodings change the difficulty of the optimization problem. As a result, GA performance strongly varies for different a, although the difficulty of f_p remains constant and is independent of a. GA performance can not accurately be predicted due to the low locality of the encodings. General statements regarding the performance (for example measured by using the schemata analysis as illustrated in Sect. 5.3.3) of binary or Gray encoding for crossover-based search are not possible as GA performance depends on the specific problem that should be solved.

5.5 Summary and Conclusions

Section 5.1 started this chapter by presenting two types of integer problems. Integer one-max problems are fully easy problems, whereas the integer fully deceptive trap is an example of a fully difficult problem. In Sect. 5.2, we presented and examined the binary, Gray and unary encoding, which are common representations for integer phenotypes. This is followed in Sect. 5.3 by a theoretical comparison of the expected performance of GEAs using the three different representations. We showed that using the non-synonymously redundant unary encoding reduces GEA performance if the optimal solution is underrepresented. Therefore, the necessary population size for solving the integer one-max and the integer deceptive trap problem is increased. Using binary encoding results in a more compact, redundancy-free representation, but the alleles are exponentially scaled. Therefore, genetic drift occurs for small and easy problems and larger population sizes are necessary. Due to the problems of the binary encoding with the Hamming cliff and its low locality, the performance of GEAs is reduced for fully easy problems, and increased for fully difficult problems. Although Gray encoding was developed to overcome the problems of the binary encoding with the Hamming cliff, an analysis of the average fitness of the schemata for the integer one-max problem shows that the Gray encoding also has low locality and preserves problem difficulty for selectorecombinative GAs less than binary encoding. Thus, the performance of selectorecombinative GAs decreases for the easy integer one-max problem and increases for the integer deceptive trap problem. To verify the theoretical prediction, we performed an empirical investigation into the performance of mutation-based and crossover-based GEAs using the different encodings in Sect. 5.4. The results confirmed the theoretical predictions.

Throughout this entire chapter, we have used the representation framework from Chap. 4 for the analysis of binary representations for integers. The analysis has shown that the pieces of representation theory can effectively be

used for predicting the performance of GEAs. We were able to explain the differences in performance of the binary, Gray, and unary representations by using the outlined theory about redundant, exponentially scaled, and low-locality representations. In particular, we gained the following insights:

We have seen that the binary encoding is exponentially scaled. However, the influence of the exponential scaling of the alleles on GEA performance can be neglected as it only affects the performance of GEAs for easy problems and small population sizes. Although the time to convergence is increased from $O(\sqrt{l})$ to $O(l)$, we can easily overcome the negative effects of the exponential scaling on GEA performance by using larger population sizes. When using the non-uniformly and non-synonymously redundant unary encoding, redundancy aspects become important. GEAs using the unary encoding fail for the integer one-max and deceptive trap as the optimal solution is strongly underrepresented for these two types of problems. Therefore, GEAs using the unary encoding perform significantly worse in comparison to GEAs using the non-redundant binary or Gray encoding. Finally, the investigation of locality reveals that both, Gray and binary encoding, have low locality. Not all neighboring genotypes correspond to neighboring phenotypes. Therefore, the influence of these representations on GEA performance is difficult to predict and depends on the specific problem that is solved. The locality of the binary encoding is worse than the Gray encoding, as for Gray-encoded genotypes there is always a neighboring genotype that corresponds to a neighboring phenotype. When using the binary encoding, it is not possible for all genotypes to reach a neighboring phenotype with one genotypic mutation.

To give a final recommendation for selectorecombinative GEAs is difficult. Both encodings, the binary and the Gray encoding, change the distances between the individuals and therefore change the complexity of the optimization problem. Thus, the resulting problem difficulty depends not only on the used representation but also on the considered optimization problem (compare Sect. 4.4.3). We have seen that some easy problems like the integer one-max problem become easier when using the binary encoding than when using the Gray encoding. However, there are other easy problems that become more difficult when using the binary encoding than when using the Gray encoding (compare the results presented in Sect. 5.4.2).

When using mutation-based GEAs instead of crossover-based GAs, the Gray encoding is the best choice (Whitley 1999). For this type of search process, Gray encoding allows a more efficient search as it better preserves the neighborhood structure (there is always a genotypic neighbor that corresponds to a phenotypic neighbor). Although the locality of Gray encoding also is not perfect (the number of genotypic neighbors is higher than the number of phenotypic neighbors), the performance of mutation-based search approaches on easy problems, and problems of bounded complexity, is higher when using Gray rather than binary encodings.

6

Analysis and Design of Representations for Trees

In the previous chapter, we illustrated that our framework modeling the influence of representations on the performance of GEAs not only works for binary phenotypes, but also for problems where the phenotypes are integers. However, it is possible to go one step further and to look at problems where the phenotypes and genotypes are completely different. One example for these types of problems are tree optimization problems. Trees are special types of graphs. Representations for trees must incorporate the additional restriction of a graph to be a tree. Therefore, if the genotypes are strings, there is a large semantic gap between tree structures (phenotypes) and strings (genotypes). In contrast to general network problems, where a representation simply has to indicate which links are used for the graph, no natural or intuitive "good" tree representations exist which are accessible for GEAs. As a result, researchers have proposed a variety of different tree representations with different properties. However, up till now no theory-based analysis exists about how GEA performance is influenced by the different types of tree representations.

The purpose of this chapter is to fill this gap and to analyze, based on the time-quality framework from Chap. 4, the influence of some of the most widely used tree representations on GEA performance. We use the existing theory about redundant, exponentially scaled, and low-locality representations to predict GEA behavior. Furthermore, the framework is used to design a new representation, the network random key (NetKey) representation. The analysis and design of direct representations, where both, genotypes and phenotypes, are trees is presented in Chap. 7. Because analyzing all known tree representations is beyond the scope of this work, we focus on some of the most widely used tree representations that assign trees to different types of strings: Prüfer numbers (Prüfer 1918), the characteristic vector encoding (Celli et al. 1995; Berry et al. 1997; Ko et al. 1997; Dengiz et al. 1997c; Dengiz et al. 1997b; Dengiz et al. 1997a; Berry et al. 1999; Premkumar et al. 2001), and the link and node biased encoding (Palmer 1994). Analyzing these representations shows that Prüfer numbers have low locality, that the redundant

characteristic vector encoding is affected by stealth mutation, and that the link and node biased encoding is not uniformly redundant.

This chapter is structured as follows. In the first section, we introduce the tree design problem and develop some basic requisites for graph problems. This is followed in Sect. 6.2 by an investigation into the Prüfer number encoding. It focuses on the Prüfer numbers' missing high locality which is necessary for GEAs to perform well on easy problems and problems of bounded difficulty. Section 6.3 presents the characteristic vector encoding, which is a redundant encoding for trees. The encoding is uniformly redundant and the performance of GEAs is independent on the structure of the optimal solution. However, GEA performance is reduced as the encoding is non-synonymously redundant. In Sect. 6.4, we show that the redundant link and node biased encoding is biased towards stars if a node-specific bias is used and biased towards the minimum spanning tree if both, link bias and node bias, are small. Finally, Sect. 6.5 presents the new network random key encoding. In analogy to random keys, the links of a tree are represented as floating numbers, and a construction algorithm constructs the corresponding tree from the keys. The NetKey representation allows us to distinguish between important and unimportant links, is uniformly redundant, uniformly scaled, and has high locality. Due to its uniform redundancy, the performance of GEAs is independent of the structure of the optimal solution. The chapter ends with concluding remarks.

6.1 The Tree Design Problem

This section provides the background for analyzing how tree representations affect GEA performance. After a brief definition of the network design problem, Sect. 6.1.2 focuses on metrics and distances for graphs. This is followed by an illustration of different tree structures like stars or lists. To be able to measure the phenotypic difficulty of a tree problem, we introduce in Sect. 6.1.4 a schema analysis for graphs. Based on the schema analysis, we present in Sect. 6.1.5 scalable test problems for graphs. The one-max tree problem, which is similar to the well known one-max problem, is a fully easy problem, whereas the deceptive trap for trees is fully difficult. Finally, the section ends with a review of former design criteria for tree encodings as provided by Palmer (1994).

6.1.1 Definitions

This subsection provides the necessary definitions for analyzing tree problems.

We define a network as a graph G with n nodes and a maximum of $n(n-1)$ links connecting the nodes. If the network is fully connected it has at least $n-1$ links. We assume that all links are undirected (they can be used in both directions) and that a network is always fully connected. Therefore, the maximum number of possible links is $n(n-1)/2$. The position of the nodes

in the graph is given a priori and the distances between two different nodes a and b are defined by using the Euclidean distance metric as

$$d_{a,b} = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}, \tag{6.1}$$

where x denotes the abscissa and y the ordinate of a node in a Cartesian coordinate system.

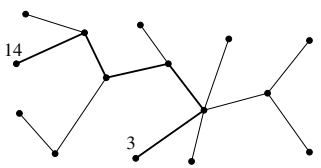


Figure 6.1. A 15 nodes tree with the path connecting nodes 3 and 14 emphasized.

The basic purpose of the network is to transport objects, for example goods or information, from some nodes in the network to other nodes. Therefore, a rule is necessary for how to transport the objects through the network. The rule for how to route the traffic through the network is based on the used routing algorithm. If the number of links in a fully connected network is larger than $n-1$, the routing of the traffic through the network can be dynamically changed dependent on the current traffic load, the delay, the failure of nodes or links, or other criteria. In contrast, if the number of links in a fully connected network is equal to $n-1$ there is only one unique path from every node to every other node and no dynamic routing is necessary.

A tree T is defined as an undirected and connected graph with no cycles. For a tree T with n nodes there are exactly $n-1$ links. It was found by Cayley (1889) that for a graph with n nodes, there are exactly n^{n-2} possible trees. A tree structure has some remarkable benefits: It represents the network structure with the lowest number of possible links to still obtain a connected graph. Furthermore, no dynamic routing is necessary as there is only one possible path for the traffic between any two nodes (compare Fig. 6.1). Finally, the size of the search space $|\Phi_{tree}| = n^{n-2}$ is much smaller than for general networks $|\Phi| \lessapprox 2^{n(n-1)/2}$.[1] However, the use of trees also has some drawbacks: Trees are very vulnerable to link or node failures. If one link or one node fails, the tree divides up into two unconnected subtrees which can not communicate with each other. However, despite this fact, trees are widely used for communication networks (Minoux 1987; Abuali et al. 1995; Elbaum and Sidi 1996; Güls 1996; Tang et al. 1997; Brittain et al. 1997; Streng 1997; Gargano et al. 1998; Gerstacker 1999; Brittain 1999; Chu et al. 1999; Chu and Premkumar 1999; Knowles et al. 1999; Grasser 2000; Gaube 2000; Edelson and Gargano

[1] We assume that there is only one possible capacity for a link. For different types of lines with k different capacities the number of possible network structure increases to $|\Phi| \lessapprox k^{n(n-1)/2}$.

2000; Edelson and Gargano 2001; Premkumar et al. 2001; Chou et al. 2001). The network design problem itself is defined as follows: Based on the

- number of network nodes n,
- locations of the n nodes,
- traffic demands between all n nodes,
- available capacities for the links,
- cost of the links dependent on the capacity and length,

we determine the

- topology (structure) of the network,
- capacity of the links,
- routing of the traffic through the network.

The general aim of the design process is to minimize the overall cost of the network with the constraint that all traffic demands between the nodes must be satisfied.

If we focus on tree structures, the capacity of the links as well as the routing of the traffic is determined by the topology. This means for trees that the optimization problem simplifies down to finding the optimal structure of the tree.

6.1.2 Metrics and Distances

As illustrated in Sect. 3.3.2, a metric is necessary for the genotypic and phenotypic space Φ_g and Φ_p to define genetic operators like mutation or recombination. The application of the mutation operator to a genotype should result in the smallest possible change in the individual, and should generate an offspring with distance 1 for the genotypes and the phenotypes. The recombination operator should ensure that the offspring inherit substructures from the parents. In terms of metric, the distance between an offspring and its parents should be lower than the distance between the two parents (compare Sect. 3.3.5).

In accordance with Chap. 3, the Hamming metric (Hamming 1980) is used for the genotypes. Thus, the Hamming distance between two binary genotypes $x^g \in \{0,1\}^l$ and $y^g \in \{0,1\}^l$ of length l is defined as

$$d_{x^g,y^g} = \sum_{i=0}^{l-1} |x_i^g - y_i^g|.$$

The distance d measures the number of alleles that are different in both individuals. Similarly, the distance between two different phenotypes (trees) is measured by using the Hamming distance d^h for trees. The Hamming distance between two trees measures the number of different links in the two trees. Therefore, the minimum Hamming distance between two different trees is $d^h = 2$.

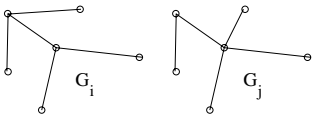


Figure 6.2. Two graphs G_i and G_j with $d_{i,j} = 1$. The Hamming distance between the two graphs is 2.

As illustrated in Fig. 6.2 the minimal Hamming distance between two trees is two, although they have $n-2$ links of all $n-1$ links in common. To simplify the metric, we define the distance $d_{i,j} \in \{0, 1, 2, \ldots n-1\}$ between two trees G_i and G_j by half of the number of different links ($d_{i,j} = \frac{1}{2} d^h_{i,j}$). It can be calculated as

$$d^p_{G_i,G_j} = d_{i,j} = \frac{1}{2} \sum_{a=1}^{n-1} \sum_{b=0}^{a-1} |l^i_{ab} - l^j_{ab}|,$$

where l^i_{ab} is 1 if the link from node a to node b exists in tree G_i and 0 if it does not exist in G_i. Then, the number of links that the two trees G_i and G_j have in common can easily be calculated as $n-1-d_{i,j}$. A mutation of a tree should result in the exchange of one link, and the distance between parent and child is $d_{parent,child} = 1$.

6.1.3 Tree Structures

When focusing on trees, different basic topological structures can be identified. In general, we can distinguish between

- stars,
- lists, and
- arbitrary trees.

Figure 6.3 illustrates the different tree types. The degree of a node is defined as the number of links which are connected to the node.

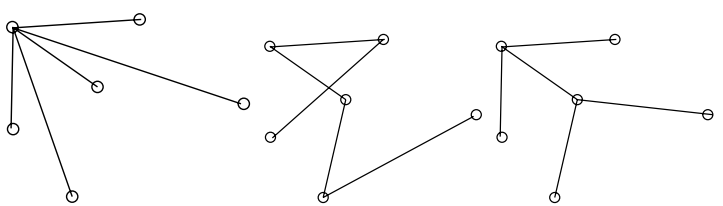


Figure 6.3. Different tree structures

A star (Fig. 6.3(a)) has one center and all other nodes are connected to the center. Therefore, the center of the network has degree $n-1$ and all other nodes have degree 1. For a network with n nodes there are n different stars.

A failure of a link or a node (except the center) disconnects only the affected node. However, if the center node fails, no further communication over the network is possible.

For a list (Fig. 6.3(b)) two nodes have degree one (leaf nodes), and all other nodes have degree 2. There are many more possible lists than stars as the number of possible lists is $\frac{1}{2}n!$. A link or a node failure results in two separate sub-lists.

Finally, there are arbitrary trees (Fig. 6.3(c)) which have no special structure except that they are trees. The degree of a node can vary from 1 to $n-1$. As for stars and lists, the sum over the degrees $deg(i)$ of all n nodes can be calculated as $\sum_{i=1}^{n} deg(i) = 2(n-1)$.

6.1.4 Schema Analysis for Graphs

In this subsection, we define schemata for graphs in analogy to schemata defined on bitstrings (compare Sect. 2.2.3). Schema analysis is helpful in determining whether graph problems are easy or difficult to solve for selectorecombinative GAs.

When assuming that GEAs process schemata, the analysis of schema fitness is the appropriate method to measure problem difficulty (see Sect. 2.3.2). The BB hypothesis (see Sect. 2.2.3) defines building blocks to be highly fit schemata of short defining length and low order. Consequently, problems are *fully easy* if all schemata of order one that contain the optimum have higher fitness than their competitors. Problems are difficult if all lower order schemata containing the global optimum are inferior to some of their competitors. The one-max problem is an example of a fully easy problem, whereas the fully deceptive trap of order k is an example for a fully difficult problem.

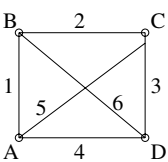


Figure 6.4. Labeling of links for $n = 4$

We can measure the difficulty of network problems for selectorecombinative GEAs by introducing schema analysis for graphs. To formally define schemata, we have to label the possible links in a graph with numbers $\{1, 2, \ldots, n(n-1)/2\}$. Figure 6.4 illustrates an example of labeling the links in a graph with $n = 4$ nodes. Then, a schema is a string of length $l = n(n-1)/2$ and the symbol at the ith position describes the existence of a link. 1 indicates that the link is established, 0 indicates no link, and * indicates don't care (dashed line). Don't care means that the link is either established or not. Figure 6.5 illustrates some possible schemata for a 4 node network using the labeling from Fig. 6.4.

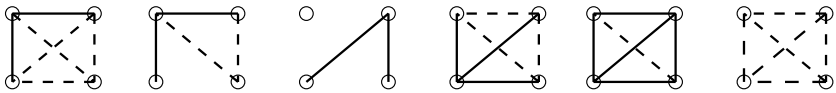


Figure 6.5. Some schemata for graphs

When using schemata for trees, there is the additional restriction that each tree has exactly $n-1$ links and it must be connected. Therefore, there must be $n-1$ ones in each solution string, and the string must encode a connected tree. This means that the average fitness of a schema must be calculated only from the trees that are represented by the schema. Other non-trees that are represented by the schema do not affect the schema fitness. This implies, for example, that schemata with more than $n-1$ ones, or more than $\frac{1}{2}n(n-1)-(n-1)$ zeros, do not exist because they do not encode a valid tree.

Using schema analysis, the difficulty of a network problem can easily be measured by the maximum order of the building blocks k. If a problem is fully easy then all lower order schemata that contain the global optimum are superior to their competitors. All building blocks have order one ($k=1$). A problem is fully deceptive if all lower order schemata that contain the optimum are inferior to their competitors. To find the optimum, GEAs must be able to find BBs of order $k=n(n-1)/2$. In general, the order of the largest BB determines the complexity of a problem. In contrast to binary strings, the length of the schemata has no meaning in the context of graphs as the labeling of the nodes does not affect problem difficulty.

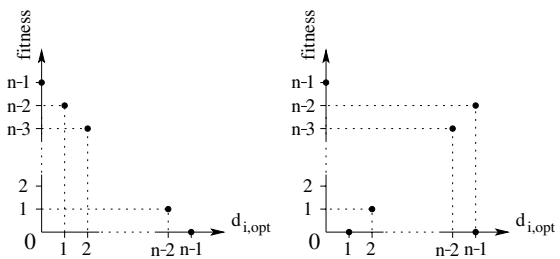
We have seen in Sect. 3.3 that low-locality representations modify problem difficulty. With the analysis of graph schemata, we can compare the difficulty of graph problems defined on the phenotypes to the difficulty of the corresponding genotypic problems defined on strings. This allows us to more easily recognize whether a tree encoding modifies the difficulty of a problem. In the following paragraphs, we consequently define a fully easy and a fully difficult scalable test problem based on the schema analysis for graphs.

6.1.5 Scalable Test Problems for Graphs

To examine the performance of optimization algorithms for the topological design of trees, standard test problems should be used. Motivated by the previous subsection, we define a fully easy and a fully difficult scalable tree problem.

The one-max tree problem is based on the integer one-max problem (compare Sect. 5.1) (Ackley 1987). An optimal solution T_{opt} is chosen either randomly or by hand. The structure of this optimal solution T_{opt} can be determined: It can be a star, a list, or a random tree with n nodes.

For the calculation of the fitness f_i of a solution T_i, the distance $d_{i,j}$ between two trees T_i and T_j is used (compare Sect. 6.1.2). Using this metric, the fitness f_i of a solution T_i depends on the distance $d_{i,opt}$ between T_i and the optimal solution T_{opt}. We can distinguish two types of problems: maximization and minimization problems. When defining a minimization problem the fitness f_i^{min} of an individual T_i is defined as the distance $d_{i,opt}$ to the optimal solution T_{opt}. Therefore, $f_i^{min} = d_{i,opt}$, where $f_i^{min} \in \{0, 1, \ldots, n-1\}$. An individual has fitness of $n-2$ if it only has one link in common with the best solution. If the two individuals do not differ ($T_i = T_{opt}$), the fitness of T_i is $f_i^{min} = 0$. If our example tree from Fig. 6.7 is chosen as the optimal solution and we have a minimization problem, the star with center D would have fitness (cost) of 1, because the two trees differ at one edge[2] ($d_{i,opt} = 1$).



(a) one-max tree problem

(b) deceptive trap tree problem

Figure 6.6. Scalable maximization test problems for trees

When defining a maximization problem, the fitness f_i^{max} of an individual T_i is defined as the number of edges it has in common with the best solution T_{opt} (compare Fig. 6.6(a)). Therefore, $f_i^{max} = n - 1 - d_{i,opt}$. If we have a maximization problem, and our example network from Fig. 6.7 is chosen as the optimal solution, the star with center D would have fitness $f^{max} = 3$ because the two networks have three links in common, and the distance between the two trees is 1.

Because both test problems (minimization and maximization problem) are similar to the standard one-max problem, they are easy to solve for mutation-based GEAs, but somewhat harder for recombination-based GAs (Goldberg et al. 1993). The existing knowledge about solving the standard integer one-max problem can be used for the one-max tree problem.

Using the schemata for trees introduced in the previous subsection shows that all building blocks of the one-max tree problem have order 1. All schemata that contain the global optimum T_{opt} are superior to their competitors. Therefore, the one-max tree problem is fully easy. For a network with 4 nodes, the schemata are already of length $l = 6$, and there are $3^6 = 729$ different schemata. Due to the limited space, we want to leave the explicit calculation

[2] A-C respectively A-D

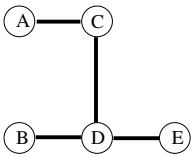


Figure 6.7. A five node tree

schema	schema fitness	represented trees	fitness of trees
		111000	3
110***	**2.33**	110100	2
		110001	2
11**1*	2	110010	2
		011100	2
01**0*	1.67	011001	2
		010101	1
		101100	2
10**0*	1.67	101001	2
		100101	1
		011010	2
01**1*	1.33	010110	1
		010011	1
		101010	2
10**1*	1.33	100110	1
		100011	1
00**0*	1	001101	1
		001110	1
00**1*	0.67	001011	1
		000111	0

Table 6.1. An example of calculating the average schema fitness for a 4 node one-max tree maximization problem where 111000 is the optimal solution. All schemata (in our example 11**0*) which contain the global optimum have higher fitness than their competitors. The problem is fully easy.

of all schema fitnesses to the reader. We only illustrate in Table 6.1 the fitness calculation for schemata where the first, second, and fifth position are fixed for a 4 node one-max tree problem. The optimal tree is defined as $x^g_{opt} = 111000$. Obviously, the schema 11**0* which contains the global optimum is superior to all its competitors.

In analogy to this fully easy one-max tree problem, we define a fully difficult deceptive trap problem for trees. As before, we choose an optimal solution T_{opt} with fitness $n-1$ (assuming a maximization problem) either by hand or randomly. Then, the fitness of all other individuals $T_i \neq T_{opt}$ is defined as $f_i = d_{opt,i} - 1$. The fitness function is illustrated in Fig. 6.6(b). This problem is fully difficult as all schemata with $k < n(n-1)/2$ containing the global optimum are inferior to their misleading competitors. Both, mutation-based and crossover-based search approaches have great problems in finding the global optimum.

Using again the example network from Fig. 6.7 as the optimal solution, a star with center D has fitness 0 because the distance to the optimal solution is 1. The optimal solution itself has fitness 4.

6.1.6 Tree Encoding Issues

We review the tree encoding issues as described by Palmer (1994) and Palmer and Kershenbaum (1994b) and relate them to the insights into the basic elements of representation theory we gained in Chap. 3. According to Palmer (compare also Sect. 2.4.3), tree representations should possess the following properties:

- A representation should be able to represent all possible trees.
- It should be unbiased in the sense that all trees are equally represented.
- A representation should be capable of representing only trees.
- The construction of the phenotype from the genotype and vice versa should be easy.
- A representation should possess locality concerning small changes.
- The schemata should encourage short, low order schemata.

We discuss these issues and relate them to the theory of representations outlined in Chap. 3 and 4.

The issue that a representation should be able to represent all possible trees is almost trivial. As long as we have no special knowledge about the problem we want to solve, it makes no sense to use a representation that might not represent some of the possible solutions. Otherwise, it could happen that GEAs search for the optimal solution, but the optimal solution can never be reached because it can not be encoded. If we have knowledge about the optimization problem, we can weaken this issue and demand representations to at least encode all the solutions we are interested in, and those which could be the optimal solution.

A representation is unbiased if all tress are represented by the same number of genotypes. Problems with biased encodings can be explained by the concept of redundant encodings illustrated in Sect. 3.1. If some phenotypes are over- or underrepresented, the encoding is biased, and the performance of GEAs is changed. The influence on the performance of GEAs by biased encodings can be modeled by using the Gambler's ruin model (Harik et al. 1999). Section 3.1 has shown that as long as the high-quality solutions are overrepresented, a bias increases performance. If the high-quality solutions are underrepresented, a decline of GEA performance is unavoidable. Therefore, with respect to a robust encoding which can be used for problems of unknown complexity, it is desirable to use unbiased encodings.

Some tree representations can also represent non-trees. These kind of representations are affected by two problems: Firstly, it could be difficult to generate valid initial populations. Secondly, the application of genetic operators can result in invalid solutions. The question arises of how to handle invalid solutions, and what to do with non-trees. In general, there are two possibili-

ties[3]: Invalid solutions can either be repaired, or they can be left unchanged in the population and hopefully they will disappear by the end of the run.

Repairing invalid solutions means that some of the trees are represented not only by valid individuals but also by some invalid solutions. Therefore, the representation is redundant. Phenotypes are not uniformly represented by the genotypes if the repair process is somehow shifted. Only a completely unbiased repair process which does not favor some tree structures guarantees an unbiased population and uniform redundancy.

To keep invalid solutions in the population could sometimes be helpful for GEAs (Orvosh and Davis 1993). Nevertheless, it must be ensured that the optimal solution at the end of the run is valid. Otherwise, the application of GEAs to tree design problems is useless as it does not result in valid solutions. To drive GEAs towards valid solutions, researchers often use penalties for invalid solutions. However, additional penalties change the fitness function and with it the behavior of GEAs. Therefore, they should be used very carefully.

An easy construction of the phenotype from the genotype, and vice versa, is necessary for an efficient implementation of GEAs. However, it depends on the complexity of the fitness function whether the computational effort for the genotype-phenotype mapping significantly affects the run duration of the computer experiments. In contrast to costly fitness evaluations, a slightly more complicated genotype-phenotype mapping could often be neglected.

The problem of locality is part of the larger question of how well the encoding preserves the complexity of a problem. As illustrated in Sect. 3.3.4, high locality guarantees that the problem difficulty remains unchanged for mutation-based search. If the locality of an encoding is low, it becomes more difficult to solve easy problems and problems of bounded difficulty.

Finally, Palmer listed Goldberg's basic design principle of meaningful building blocks (compare Sect. 2.4.1) and demanded encodings to encourage short, low order schemata. Otherwise, "long schemata cause genetic algorithms to drift" (Palmer 1994, p. 40). However, as illustrated in Sect. 3.2, drift is caused by non-uniformly scaled alleles and domino convergence, and not by the length and the size of the building blocks. The size and length of the building blocks determine the complexity of a problem for selectorecombinative GAs.

We recognize that the tree design issues from Palmer can be well understood by using the framework presented in Chap. 4.

6.2 Prüfer Numbers

Prüfer numbers are a widely used representation for trees. The purpose of this section is to use the framework from Chap. 4 for an investigation into the

[3]Of course, there is a third possibility: To remove the individual from the population. However, we do not consider this case.

properties of Prüfer numbers. The analysis focuses on the low locality of the encoding and shows how GEA performance is affected.

The section starts with an historical review of the use of the Prüfer number encoding in the context of genetic and evolutionary search. The review shows a strong increase in interest into the encoding over the last 5 to 10 years. This is followed by the construction and deconstruction process of Prüfer numbers. Section 6.2.3 illustrates the benefits and drawbacks of the encoding. The use of the Prüfer number encoding is very charming due to its advantageous properties, although the low locality of the encoding has already been identified by Palmer (1994) to be its main drawback. Subsequently, in Sect. 6.2.4 we present a deeper investigation into how exactly the low locality damages the performance of GEAs. In analogy to Sect. 3.3, we illustrate why high locality is necessary for an encoding to preserve problem difficulty and perform random walks through the search space. After an analysis of the neighborhood structure of Prüfer numbers, we finally present empirical results for different tree structures using mutation and recombination-based evolutionary search methods. The section ends with concluding remarks.

6.2.1 Historical Review

We give a brief historical review of the development and use of the Prüfer number encoding in the context of GEAs.

Cayley (1889) identified the number of distinct spanning trees on a complete graph with n nodes as n^{n-2} (Even 1973, pp. 103-104). Later, this theorem was very elegantly proven by Prüfer (1918) by the introduction of a one-to-one correspondence between spanning trees and a string of length $n-2$ over an alphabet of n symbols. This string is denoted as Prüfer number, and the genotype-phenotype mapping is the Prüfer number encoding. It is possible to derive a unique tree with n nodes from the Prüfer number of length $n-2$ and vice versa (Even 1973, pp. 104-106). Of course there are other one-to-one mappings from strings of $n-2$ labels onto spanning trees on the n labeled links. One example is the Blob Code which was developed and proposed by Picciotto (1999). Julstrom (2001) compared this encoding to Prüfer numbers and found for easy problems a higher performance of GEAs using the Blob Code than Prüfer numbers.

Later, in the context of GEAs, several researchers used the Prüfer number encoding for the representation of trees. Palmer used the encoding in his doctoral thesis at the beginning of the nineties (Palmer 1994; Palmer and Kershenbaum 1994a; Palmer and Kershenbaum 1994b), and compared the performance of Prüfer numbers with some other representations for the optimal communication spanning tree problem. However, he noticed that the Prüfer number encoding has low locality and therefore is not a good choice for encoding trees. The low performance of the encoding was confirmed by Julstrom (1993) who used Prüfer numbers for the rectilinear Steiner problem, and also observed low GEA performance using this encoding.

About the same time, Abuali et al. (1994) used Prüfer numbers for the optimization of probabilistic minimum spanning trees (PMST) with GEAs. The investigation focused more on the influence of different operators than on the performance of Prüfer numbers. However, at the end of the work, the conclusion was drawn that in contrast to Palmer and Julstrom, Prüfer numbers "lead to a natural GEA encoding of the PMST problem" (Abuali et al. 1994, p. 245). Some years later, similar results were reported by Zhou and Gen (1997) who successfully used the Prüfer encoding for a degree constraint minimum spanning tree problem. The degree constraint was considered by repairing invalid solutions that violate the degree constraints. Furthermore, Prüfer numbers were used for spanning tree problems (Gen et al. 1998; Gen et al. 1998), the time-dependent minimum spanning tree problem (Gargano et al. 1998), the fixed-charge transportation problem (Li et al. 1998) and a bicriteria version of it (Gen and Li 1999), and a multi-objective network design problem (Kim and Gen 1999). Most of this work reported good results when using Prüfer numbers, and labeled the encoding to be (very) suitable for encoding spanning trees. As an example of positive results we want to cite Kim and Gen (1999), who wrote:

> "The Prüfer number is very suitable for encoding a spanning tree, especially in some research fields, such as transportation problems, minimum spanning problems, and so on."[4]

However, other relevant work by Krishnamoorthy et al. (1999), who used Prüfer numbers for the degree constraint spanning tree problem, from Julstrom (2000) who compared a list of edges encoding with Prüfer numbers, or from Gottlieb and Eckert (2000) who used Prüfer numbers for the fixed charge transportation problem showed that Prüfer numbers result in a low GEA performance. A summarizing study by Gottlieb et al. (2001) compared the performance of Prüfer numbers for four different network problems and concluded that Prüfer numbers always perform worse than other encodings, and are not suitable for encoding trees when using GEAs.

To explain the differences between the good and bad results obtained by GEAs using Prüfer numbers, Rothlauf and Goldberg (1999) investigated the locality of the encoding more closely. It was shown that Prüfer numbers only have high locality if they encode stars. For all other tree types the locality is low which leads to a degradation of GEAs (see also Rothlauf and Goldberg (2000) and Rothlauf et al. (2001)). Therefore, the differences in performance could be well explained if one assumes that the performance of GEAs depends on the structure of the optimal solution. Obviously, researchers who report good solutions when using Prüfer numbers used problems where the optimal solution is more star-like and therefore easy to find for GEAs. However, when using Prüfer numbers for more general, non-star like problems, a strong decrease in GEA performance is inescapable. The results from Roth-

[4]Special thanks to Bryant A. Julstrom for his help with finding this statement.

lauf and Goldberg (1999) were confirmed by Gottlieb and Raidl (2000) who investigated the effects of locality on the dynamics of evolutionary search.

We have seen that the performance of GEAs using Prüfer numbers is a strongly discussed topic. Some researchers report good results and favor the use of Prüfer numbers. Other researchers, however, point to the low locality of the encoding, report worse results and advise us not to use Prüfer numbers. A closer investigation into how locality depends on the structure of the tree could solve these contradictory results. As the work from Rothlauf and Goldberg (2000) indicates that the locality of Prüfer numbers strongly depends on the structure of the tree, GEAs show good results if the good solutions are star-like, and worse results for all other types. In Sect. 6.2.4, we review the main results from Rothlauf and Goldberg (1999) and Rothlauf and Goldberg (2000) and extend it with additional work.

6.2.2 Construction

We review the construction rule for the Prüfer number encoding. We present both sides of the story: How a Prüfer number can be constructed from a tree, and how a tree can be constructed from a Prüfer number.

The Construction of the Prüfer Number from a Tree

The degree $deg(i)$ of a node i denotes the number of links that are connected to the node. Thus, as a fully connected tree has exactly $n-1$ links, the degree $deg(i)$ of a node i lies between 1 and $n-1$. A node has degree one if it is a leaf node. It has degree $n-1$ if it is the center of a star. There are always at least two nodes which have degree 1.

The Prüfer number itself encodes an n-node tree with a string of length $n-2$, and each element of the string is of base n. As the mapping is one-to-one, a Prüfer number is a unique encoding of a tree, and there are n^{n-2} different possible Prüfer numbers (Cayley 1889; Prüfer 1918).

For the construction of the Prüfer number from a tree, we label all nodes with numbers from 1 to n. Then, the Prüfer number can be constructed from a tree by the following algorithm:

1. Let i be the lowest numbered node of degree 1 in the tree ($deg(i) = 1$).
2. Let j be the one node which is connected to i (there is exactly one). The number of the jth node is the furthest right digit of the Prüfer number.
3. Remove node i and the link (i, j) from the tree and from further consideration.
4. Go to 1 until only two nodes (that means one link) are left.

After termination of the construction rule, we have a Prüfer number with $n-2$ digits which represents the tree. An efficient implementation of this algorithm uses a priority queue implemented in a heap to hold the nodes of degree 1. The algorithm's time complexity is then $O(n \log n)$.

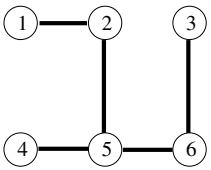


Figure 6.8. A tree and the corresponding Prüfer number $P = 2565$

Let us demonstrate the construction of the Prüfer number with a brief example. The network in Fig. 6.8 has 6 nodes. Therefore, the Prüfer number consists of 4 digits. The lowest numbered node with degree 1 is node 1. This node is connected to node 2 so the Prüfer number starts with a 2. We remove node 1 from further consideration and search for the lowest numbered node with degree 1. We identify node 2 which is connected to node 5. The Prüfer number becomes 25. After removing node 2, node 3 is the lowest numbered node which is eligible (it has degree 1). Node 3 is connected to 6 so we get 256. The node 4 is the lowest eligible node and we add 5 to the Prüfer number. Finally, only two nodes remain in the tree. The algorithm stops and the resulting Prüfer number is 2565.

The Construction of the Tree from the Prüfer Number

The construction of the tree from the Prüfer number follows the construction of the Prüfer number from the tree. It goes as follows:

1. Let P be a Prüfer number with $n-2$ digits. All node numbers which are not in P can be used for the construction of the tree (are eligible).
2. Let i the lowest numbered eligible node. Let j be the leftmost digit of P.
3. Add the link (i, j) to the tree.
4. Designate i as no longer eligible and remove the leftmost digit j from the Prüfer number.
5. If j does not occur anywhere else in the remaining Prüfer number, designate j as eligible.
6. Go to 2 until no digits remain in the Prüfer number. If no digits are left, then there are exactly two numbers, r and s, which are eligible. Finally, add the link (r, s) to the tree.

We also illustrate this construction rule with a brief example. We want to construct the tree from the Prüfer number $P = 2565$. Eligible nodes are 1, 3 and 4. As 1 is the lowest eligible node, and 2 is the leftmost digit of the Prüfer number, we add the link $(1, 2)$ to the tree. 1 is then no longer eligible, and 2 does not occur anywhere else in the string. Therefore, the nodes 2, 3 and 4 are eligible and P becomes 565. Now, 2 is the lowest eligible node and we add the link $(2, 5)$ to the tree. As 5 occurs somewhere else in the string, we do not designate 5 as eligible. Thus, we only remove 2 from our pool of eligible numbers, and then we can add the link $(3, 6)$ to the tree. Now, only the nodes 4 and 6 are eligible and $P = 5$. We continue with adding $(4, 5)$. Finally, all

digits are removed from P and the numbers 5 and 6 remain eligible. The link $(5, 6)$ is added to the tree and the algorithm terminates. We have constructed the tree illustrated in Fig. 6.8.

6.2.3 Properties

We analyze the properties of the Prüfer number encoding by using the design issues from Palmer and Kershenbaum (1994a) and Palmer (1994) (see Sect. 6.1.6). Furthermore, we relate these properties to the more general properties of representations we developed in Chap. 3.

Benefits

The Prüfer number encoding is a very elegant and interesting encoding with some remarkable benefits:

- Every tree can be represented by a Prüfer number.
- Only trees are represented by Prüfer numbers.
- Every Prüfer number represents exactly one tree.
- All trees are represented uniformly (unbiased).

A look at the construction rule of the Prüfer number shows that a Prüfer number is able to represent all possible trees. Because every tree has at least two nodes with degree 1, the construction rule can be applied to every tree. The user should notice that the original intent of the Prüfer number was to prove Cayley's theorem (Cayley 1889) by introducing Prüfer numbers. It was also shown by Prüfer (1918) that Prüfer numbers only represent trees. Therefore, a Prüfer number can be randomly created and it always represents a tree. In contrast to many other representations, no repairing of a randomly chosen individual is necessary. Furthermore, it is also not necessary to repair individuals that are generated by genetic operators in each generation. The first three benefits of the Prüfer numbers can be summarized by denoting the Prüfer number encoding as a one-to-one mapping. The mapping is not only surjective, but also bijective.

One consequence of a one-to-one mapping is that all trees are uniformly represented as each tree is represented by exactly one specific Prüfer number. The number of different trees for a graph with n nodes is n^{n-2}, and there are also exactly n^{n-2} different Prüfer numbers for an n node tree. Therefore, GEAs using Prüfer numbers have no problems with redundancy. GEAs using Prüfer numbers can not be affected by the over- or underrepresentation of some individuals.

These advantages make Prüfer numbers an interesting encoding for trees. However, the use of Prüfer numbers is connected to some serious drawbacks.

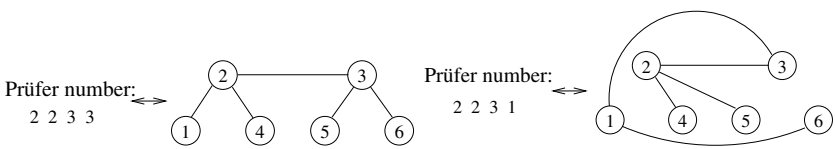
Drawbacks

The Prüfer number has the disadvantages of

- complex calculation and
- low locality.

In comparison to some other representations, the construction of the Prüfer number is more complex and not straightforward. But, it can be done using the help of a heap in $O(n \log n)$. This seems to be acceptable for most problems.

The most important disadvantage of the Prüfer number is the low locality of the representation. Small changes in the Prüfer number string can lead to large changes in the represented network. This means, the mapping from the phenotype to the genotype is not homogeneous. Therefore, the basic mutation operator that searches the local solution space around an individual does not generate offspring that are similar to their parents. A descendant does not inherit the important properties of its parents. Thus, mutation works not as a local search, but more as a random search over the solution space (compare Sect. 3.3.4).



(a) A tree and its Prüfer number $P = 2233$

(b) A tree and its Prüfer number $P = 2231$

Figure 6.9. The low locality of the Prüfer number encoding. A change of one digit changes three links in the corresponding tree.

A small example illustrates the low locality of the encoding. Changing the last digit in the Prüfer number of Fig. 6.9(a) from 3 to 1 yields 2231, which decodes to the links (2,4), (2,5), (3,2), (1,3), and (1,6) (compare Fig. 6.9(b)). Only two of the original tree's five links exist in the offspring.

6.2.4 The Low Locality of the Prüfer Number Encoding

As illustrated in the previous subsection, the Prüfer number encoding is affected by low locality. The purpose of this subsection is to investigate the locality of the encoding more closely. The locality of Prüfer numbers is examined by performing two different investigations: Firstly, we perform random walks through the search space and examine the distances between parents and offspring. Secondly, we investigate the neighborhood of the genotypes and phenotypes. We examine the locality of the neighboring individuals, and determine their number. Finally, we present an empirical verification of the theoretical predictions for mutation- and crossover-based evolutionary search.

Random Walks

We present experiments for evaluating the locality of the Prüfer number encoding. For this purpose we perform random walks through the genotypic and phenotypic search space and analyze the resulting changes in the corresponding phenotypes/genotypes.

Figure 6.10 shows the encoding of a tree $x^p \in \Phi_p$ as a Prüfer number $x^{g1} \in \Phi_{g1}$ and the encoding of the Prüfer number as a bitstring $x^g \in \Phi_g$. The genetic operators are applied to the genotypes x^g. The Prüfer number itself is a sequence of integers and is represented as a bitstring using the binary encoding (compare Sect. 5.2). Therefore, the mapping from the bitstring to the Prüfer number $f_g : \Phi_g \rightarrow \Phi_{g1}$ is affected by scaling and has the properties discussed in Sect. 5.3.3. The mapping from the Prüfer numbers to the trees $f_{g1} : \Phi_{g1} \rightarrow \Phi_p$ is described in Sect. 6.2.2. Notice that for a tree with n nodes, the Prüfer number has $n-2$ digits, and the bitstring $(n-2)\lceil \log_2(n) \rceil$ bits.

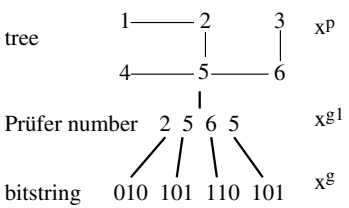
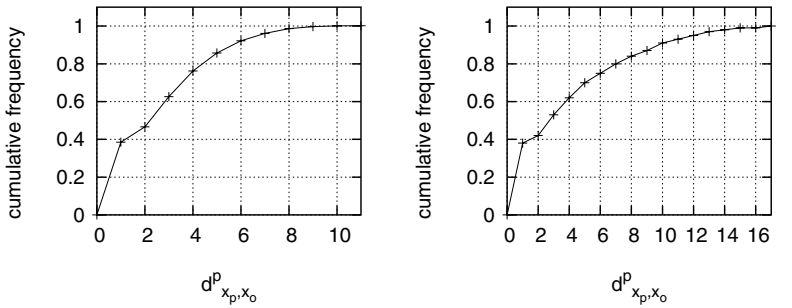


Figure 6.10. A tree, its Prüfer number and the corresponding bitstring

The locality of the Prüfer number encoding can be measured by performing a random walk through one of the solution spaces Φ_g, Φ_{g1}, or Φ_p, and measuring the distances between parent x_p and offspring x_o in the other two solution spaces. A random walk through a search space Φ is defined by performing iteratively small changes. Therefore, the distance between parent x_p and offspring x_o in the solution space we are performing our random walk in is $d_{x_p,x_o} = 1$. According to Fig. 6.10 there are three different possibilities:

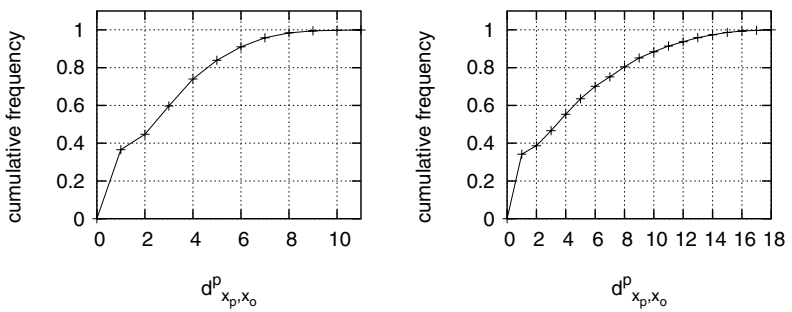
1. A random walk through Φ_g (a step changes one bit of the bitstring, $d^g_{x_o,x_p} = 1$).
2. A random walk through Φ_{g1} (a step changes one digit of the Prüfer number, $d^{g1}_{x_o,x_p} = 1$).
3. A random walk through Φ_p (a step changes one edge in the tree, $d^p_{x_o,x_p}$=1).

A random walk through Φ_g means randomly changing one bit of x^g and examining how many links change in the corresponding tree x^p. Furthermore, the change of one bit in the bitstring x^g results in the change of exactly one digit of the Prüfer number x^{g1}. A random walk through Φ_{g1} means randomly changing one digit of the Prüfer number x^{g1} and measuring how many links are different in the resulting x^p. Notice that the change of one digit in x^{g1} results in up to $\log_2(n)$ different bits in x^g. A random walk through Φ_p means



(a) $n = 16$ (b) $n = 32$

Figure 6.11. Distribution of phenotypic distances $d^p_{x_p,x_o}$ for neighboring bitstrings $x_p, x_o \in \Phi_g$ on 16 and 32 nodes. We perform a random walk through Φ_g ($d^g_{x_p,x_o} = 1$), and show how many links are different if one bit of the bitstring is changed.

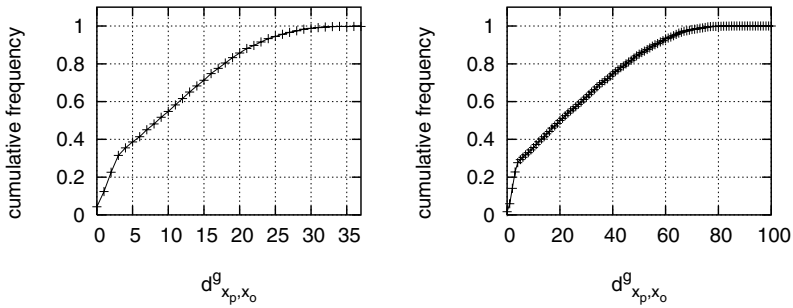


(a) $n = 16$ (b) $n = 32$

Figure 6.12. Distribution of phenotypic distances $d^p_{x_p,x_o}$ for neighboring Prüfer numbers $x_p, x_o \in \Phi_{g1}$ on 16 and 32 nodes. We perform a random walk through Φ_{g1} ($d^{g1}_{x_p,x_o} = 1$), and show how many links are different if one digit of the Prüfer number is changed.

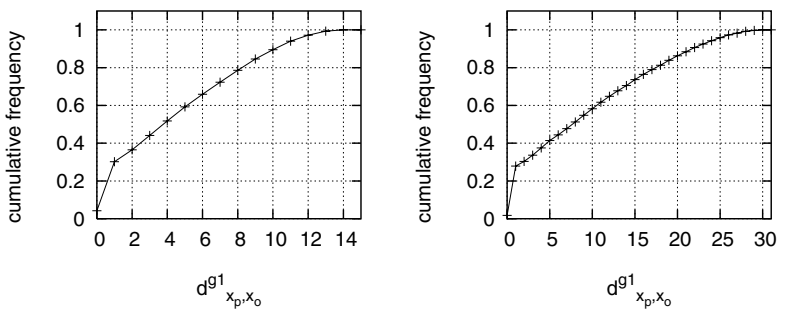
that one link of the tree x^p is replaced by a randomly chosen link, and the difference of bits/digits in the bitstring/Prüfer number is examined.

Figures 6.11, 6.12, 6.13, and 6.14 present the results of the random walks. We show the cumulative frequency over the distances d_{x_p,x_o} between the parent x_p and the offspring x_o. The phenotypic distance between the parent x_p and offspring x_o is denoted as $d^p_{x_p,x_o}$ (for trees $x_p, x_o \in \Phi_p$), as $d^{g1}_{x_p,x_o}$ (for Prüfer numbers $x_p, x_o \in \Phi_{g1}$), or as $d^g_{x_p,x_o}$ (for bitstrings $x_p, x_o \in \Phi_g$). In all our experiments, the start individual for the bitstring, the Prüfer number, or the tree is chosen randomly. To gain statistically significant results independently of the start individual, 400 steps (mutations) were carried out in each of the 20 runs. Thus, we performed overall 8,000 steps in the search space.



(a) $n = 16$ (b) $n = 32$

Figure 6.13. Distribution of genotypic distances $d^g_{x_p,x_o}$ for neighboring trees $x_p, x_o \in \Phi_p$ on 16 and 32 nodes. We perform a random walk through Φ_p ($d^p_{x_p,x_o} = 1$), and show how many bits are different if one link of the tree is changed.



(a) $n = 16$ (b) $n = 32$

Figure 6.14. Distribution of genotypic distances $d^{g1}_{x_p,x_o}$ for neighboring trees $x_p, x_o \in \Phi_p$ on 16 and 32 nodes. We perform a random walk through Φ_p ($d^p_{x_p,x_o} = 1$), and show how many digits of the Prüfer number are different if one link of the tree is changed.

The results for a random walk through Φ_g (Fig. 6.11) and Φ_{g1} (Fig. 6.12) show that only about 40% of the one bit/digit changes lead to a change of one link in the tree ($d^p_{x_p,x_o} = 1$). More than 35% (16 nodes) or 50% (32 nodes) of all one bit/digit changes result in trees x^p_o with at least four different links ($d^p_{x_p,x_o} \geq 4$). Therefore, the locality of the mapping from the genotype (bitstring as well as Prüfer number) to the phenotype is low. Low genotypic distances ($d^g_{x_p,x_o} = 1$ or $d^{g1}_{x_p,x_o} = 1$) do not correspond to low phenotypic distances $d^p_{x_p,x_o}$.

When walking through the phenotypic solution space (trees) the plots in Fig. 6.13 show that only about 50% of all one link changes result in a change of less than eight bits (16 nodes), respectively 20 bits (32 nodes) in the bitstring.

For the Prüfer number (Fig. 6.14) about 75% of the neighboring trees are different in more than one digit. Therefore, the locality of the phenotype-genotype mapping is also low.

The random walks through Φ_g and Φ_p have shown that the locality of the Prüfer number representation is low. Most of the small steps in the phenotypic and genotypic search space result in unacceptably high changes in the corresponding genotypic and phenotypic search space. The following paragraphs investigate whether the locality of the Prüfer numbers encoding is uniformly low everywhere in the search space, or if there are some areas of high locality.

Analysis of the Neighborhood

Performing random walks through the different search spaces has revealed that the locality of the Prüfer number encoding is low. Therefore, we investigate whether the locality of the encoding is uniformly low, or if there are differences in locality for different areas of the search space. The search space can be separated into different areas by making assumptions about the structure of the represented tree such as being a star or a list.

To investigate whether the locality of the Prüfer number encoding is different for different areas of the search space, we choose an individual x with specific properties and examine its locality. We examine all individuals y with distance $d_{x,y} = 1$ and measure the resulting genotypic or phenotypic distance. As an individual is defined to be a neighbor to another individual if the distance between the two individuals is 1, our examination is nothing more than an examination of the neighborhood of specific individuals. In analogy to the random walks, we investigate the neighborhood of individuals in all three search spaces Φ_g, Φ_{g1} and Φ_p:

- Neighborhood of an individual $x^g \in \Phi_g$ (all neighbors $y^g \in \Phi_g$ that are different in one bit from the examined individual, $d^g_{x^g,y^g} = 1$).
- Neighborhood of a Prüfer number $x^{g1} \in \Phi_{g1}$ (all neighbors $y^{g1} \in \Phi_{g1}$ that differ in one digit, $d^{g1}_{x^{g1},y^{g1}} = 1$).
- Neighborhood of a tree $x^p \in \Phi_p$ (all neighbors $y^p \in \Phi_p$ that have distance $d^p_{x^p,y^p} = 1$).

We examine the complete neighborhood of an individual either in Φ_g, Φ_{g1}, or Φ_p and measure the corresponding distances in the two others. This investigation can be performed for four different types of networks:

(i) Star: One node is of degree $n - 1$ and the rest of the nodes have degree 1.
(ii) Random list: Two nodes are of degree 1 (the first and the last node of the list) and all other nodes have degree 2. The numbering of the nodes is random.

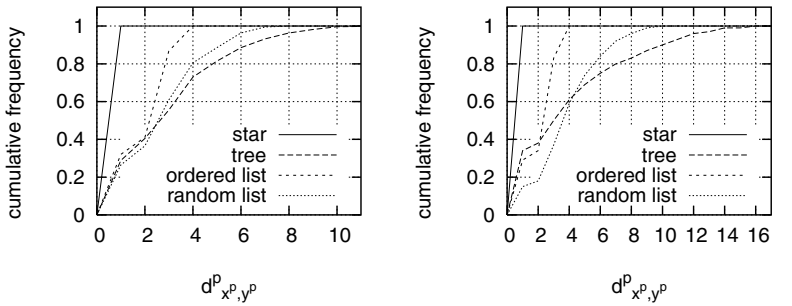
(iii) Ordered list: Like random list, but the nodes in the list are connected in ascending order. Node k is connected to $k+1$, node $k+1$ is connected to $k+2$ and so on. If the highest numbered node n is not a leaf node, then it is connected to node 1.
(iv) Tree: An arbitrary tree.

We distinguish between ordered and random lists because the locality of the Prüfer number encoding is slightly different for ordered and random lists.

Figures 6.15, 6.16, 6.17, and 6.18 examine the neighborhood of stars, random and ordered lists, and arbitrary trees with 16 and 32 nodes. A bitstring representing a tree has either length $l = 56$ (16 nodes) or $l = 150$ (32 nodes); a Prüfer number encoding a tree has either 14 (16 nodes) or 30 (32 nodes) digits. For every problem instance, the complete neighborhood of 1000 randomly chosen individuals x is examined. Figure 6.15 shows distributions of phenotypic distances $d^p_{x^p,y^p}$ between a randomly chosen bitstring x^g and all neighboring y^g, where $d^g_{x^g,y^g} = 1$. Figure 6.16 shows distributions of phenotypic distances $d^p_{x^p,y^p}$ between a randomly chosen Prüfer number x^{g1} and all neighboring y^{g1}, where $d^{g1}_{x^{g1},y^{g1}} = 1$. Figures 6.17 and 6.18 show distributions of genotypic distances ($d^g_{x^g,y^g}$ and $d^{g1}_{x^{g1},y^{g1}}$) between a randomly chosen tree x^p and all neighboring graphs y^p, where $d^p_{x^p,y^p} = 1$; that is, for spanning trees that differ in one link.

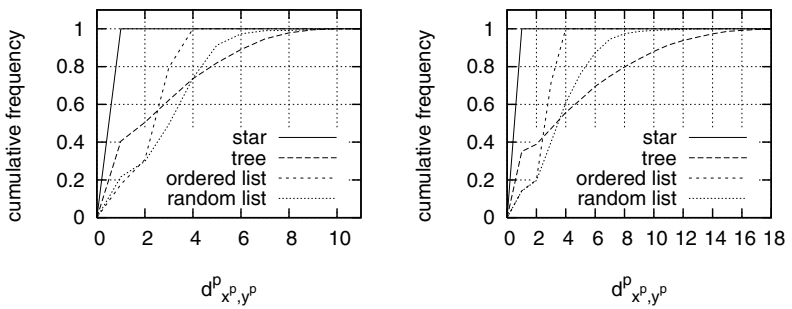
Figures 6.15 and 6.16 reveal that the neighborhood of a bitstring-encoded tree, as well as a Prüfer number representing a spanning tree, depends on the structure of the encoded tree. If the bitstring/Prüfer number encodes a star, all genotypic neighbors also have a phenotypic distance of one. This means that the locality of the bitstring/Prüfer number is perfect for stars. If the bitstring/Prüfer number encodes an ordered list, the genotypic neighbors have a maximum phenotypic distance of 4, independently of the number of nodes. This means, a change of one bit/digit of x^g or x^{g1} that encode an ordered list results in a maximum phenotypic distance $d^p_{x^p,y^p} = 4$. In contrast, bitstrings and Prüfer numbers that encode random lists or random trees, show low locality, and most of the genotypic neighbors are phenotypically completely different.

The neighborhoods of trees are examined in Figs. 6.17 and 6.18. Similarly to the genotypic neighborhood, all neighbors of a star have a genotypic distance in the Prüfer number space of one ($d^{g1}_{x^{g1},y^{g1}} = 1$ illustrated in Fig. 6.18). However as a bitstring y^g encodes a Prüfer number y^{g1} using a binary encoding, up to $\lceil \log_2(n) \rceil$ bits are changed in the bitstring y^g that represents the Prüfer number y^{g1} (Fig. 6.17). The change of one digit in a Prüfer number can result in a change of up to $\lceil \log_2(n) \rceil$ bits in the bitstring. Therefore, the locality of the Prüfer number encoding is high for stars. For arbitrary trees and lists, the change of one link often results in a completely different bitstring/Prüfer number. Therefore, the locality of the Prüfer number encoding is low for trees and lists.



(a) 16 nodes (b) 32 nodes

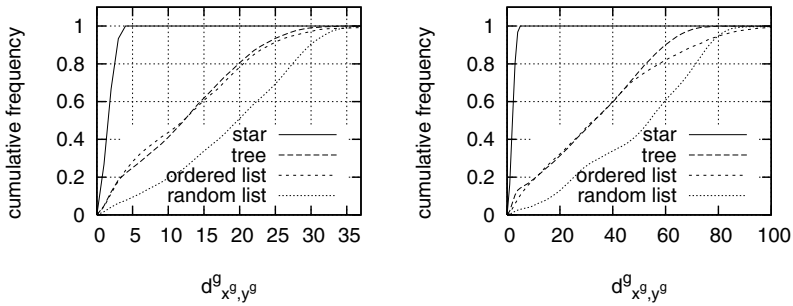
Figure 6.15. Distribution of phenotypic distances $d^p_{x^p,y^p}$, where $\{y^g | d^g_{x^g,y^g} = 1\}$. The graphs illustrate how many links are different in the tree y^p when examining the complete neighborhood of a randomly chosen genotypic bitstring x^g.



(a) 16 nodes (b) 32 nodes

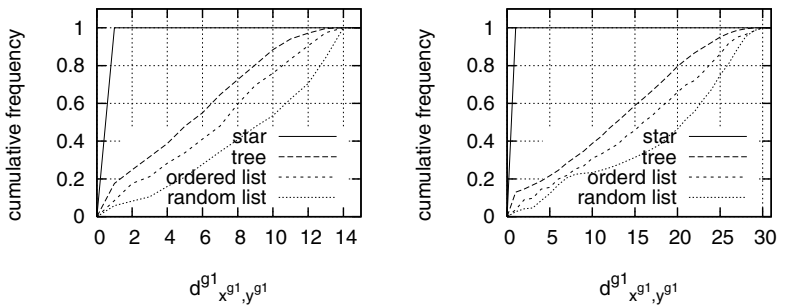
Figure 6.16. Distribution of phenotypic distances $d^p_{x^p,y^p}$, where $\{y^{g1} | d^{g1}_{x^{g1},y^{g1}} = 1\}$. The graphs illustrate how many links are different in the tree y^p when examining the complete neighborhood of a randomly chosen genotypic Prüfer number x^{g1}.

The results show that the locality of the Prüfer number encoding is highly irregular and depends on the phenotypic structure of the encoded tree. If a Prüfer number encodes a list or arbitrary tree, the locality of the encoding is low. Then, most of the genotypic neighbors of a Prüfer number-encoded genotype are phenotypically completely different. However, if Prüfer numbers encode stars, the locality of the encoding is perfect. All genotypic neighbors of a star are also phenotypic neighbors. These results raise two new questions: Why do Prüfer numbers which encode stars have high locality? How large are the areas of high locality? We answer these questions in the following paragraphs.



(a) 16 nodes (b) 32 nodes

Figure 6.17. Distribution of genotypic distances $d^g_{x^g,y^g}$, where $\{y^p | d^p_{x^p,y^p} = 1\}$. The graphs illustrate how many bits are different in y^g when examining the complete neighborhood of a randomly chosen tree x^p.



(a) 16 nodes (b) 32 nodes

Figure 6.18. Distribution of genotypic distances $d^{g1}_{x^{g1},y^{g1}}$, where $\{y^p | d^p_{x^p,y^p} = 1\}$. The graphs illustrate how many digits are different in the Prüfer number y_{g1} when examining the complete neighborhood of a randomly chosen tree x^p.

Number of Neighbors

We have seen that Prüfer numbers only have high locality if they encode stars. Therefore, we focus on two issues: Firstly, we want to reveal why Prüfer numbers have high locality when encoding stars. And secondly, we want to know how large the areas are of high locality. Finding answers for these questions helps us to more accurately predict the behavior of GEAs for different tree optimization problems. The investigation will show that the number of neighbors has a major impact on the answers to both questions.

The previous investigations have shown that some Prüfer numbers have high locality. A Prüfer number representing a star has perfect locality ($d_m = 0$) because all phenotypic neighbors of a star are also genotypic neighbors.

To shed light on the question of why exactly stars have perfect locality, we calculate the number of neighbors for both, Prüfer numbers and trees.

A Prüfer number x^{g1} uses $n-2$ digits of base n to encode a tree x^p with n nodes. Because we can change each of the $n-2$ digits to $n-1$ different integers, each Prüfer number has exactly $(n-1)\times(n-2)$ neighbors. Furthermore, each Prüfer number x^{g1} with $n-2$ digits is encoded as a bitstring x^g of length $(n-2)\times\lceil\log_2(n)\rceil$. So each bitstring x^g has $(n-2)\times\lceil\log_2(n)\rceil$ neighbors. A change of one digit in the Prüfer number x^{g1} can result in up to $\lceil\log_2(n)\rceil$ different bits in x^g. As we are mainly interested in the Prüfer number encoding, we want to focus in the following paragraphs only on Prüfer strings (integers) and neglect the encoding of Prüfer strings as bitstrings. The reader should notice that the number of neighbors of a Prüfer number is independent of the structure of the encoded tree.

A star on n nodes has $(n-1)(n-2)$ neighbors obtained by replacing one of its links with another feasible link. Therefore, for stars the number of neighbors is the same for the phenotypes and for the genotypes. Furthermore, a star's neighbors are represented by the neighbors of its Prüfer number obtained by changing one of the number's symbols; as already mentioned, these neighbors number also $(n-1)(n-2)$. For stars, the genotypic and phenotypic neighborhoods coincide, and therefore locality is maximal.

This seems auspicious, but tree localities vary with the shape of the tree. A list is a spanning tree with two leaves and $n-2$ nodes of degree 2. In a list's Prüfer number, all the symbols are distinct, and each Prüfer number has, as already mentioned, $(n-1)(n-2)$ neighbors. However, a list on n nodes has $\sum_{i=1}^{n-1} i(n-i)-1 = n\sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} i^2 - (n-1) = \frac{1}{2}n^2(n-1) - \frac{1}{6}n(n-1)(2n-1) - n + 1 = \frac{1}{6}n(n-1)(n+1) - n + 1$ neighbors (Gerstacker 1999). Therefore, for lists the number of phenotypic neighbors is much higher than the number of genotypic neighbors. Stars and lists have the smallest and largest phenotypic neighborhoods, respectively. All other spanning trees fall between these extremes which Fig. 6.19 plots as a function of the number n of nodes.

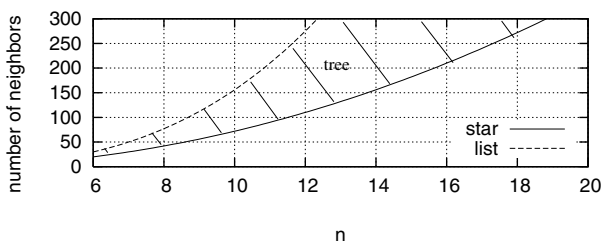


Figure 6.19. Phenotypic neighborhood sizes for lists and stars, as functions of the number of nodes. The values for all other trees lie between these curves.

We see that the number of neighbors of a phenotype increases when modifying a star towards a list. However, the number of neighbors of a Prüfer number remains constant and is independent of the structure of the encoded

tree. So there is a mismatch between the number of neighbors of non-star trees and of Prüfer numbers. Therefore, the locality of all Prüfer numbers not encoding a star could not be perfect as phenotypes always have a higher number of neighbors than genotypes.

The results concerning the number of neighbors are summarized and illustrated for some example networks in Table 6.2. The number of neighbors for arbitrary trees is between the number for stars and lists and must be separately calculated for each tree. It depends on the degrees of the nodes in the tree.

Table 6.2. Number of neighbors for graphs, Prüfer numbers, and bitstrings

nodes		graph x^p	Prüfer number x^{g1}	bitstring x^g
n	star	$(n-1)(n-2)$	$(n-1)\ (n-2)$	$(n-2) \times \lceil \log_2(n) \rceil$
	list	$\frac{1}{6}n(n-1)(n+1)-n+1$		
8	star	42	42	18
	list	77		
	tree	62.45 (avg.)		
16	star	210	210	56
	list	665		
	tree	447.2 (avg.)		
32	star	930	930	150
	list	5425		
	tree	2595 (avg.)		

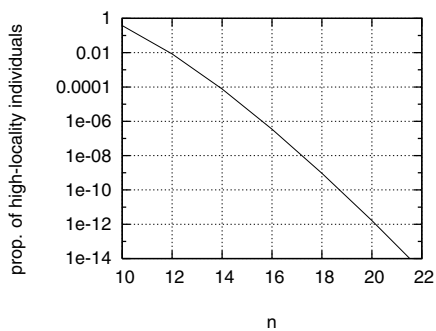


Figure 6.20. The proportion of spanning trees on n nodes whose Prüfer numbers have high locality, defined as differing from the Prüfer number of a star in no more than $i_{\max} = 5$ digits.

After we have explained why Prüfer numbers can only have high locality when encoding stars, we focus on the question of how large the areas of

high locality are. GEAs only search effectively in regions near stars. In these areas, the locality is high and the encoding allows a guided search. To approximate the number of individuals with high locality, we extend the definition of neighbors to include trees whose Prüfer numbers x^{g1} differ in at the most $i_{\max}$ digits ($i_{\max} \ll n$). This means, we assume an individual x^{g1} to have high locality if its distance $d^{g1}_{x^{g1},star}$ towards a star is equal or lower than $i_{\max}$. The number of individuals x^{g1} which have the maximum distance $i_{\max}$ towards a star ($d^{g1}_{x,star} \leq i_{max}$) can be calculated as $\sum_{i=0}^{i_{\max}} \binom{n-2}{i}(n-1)^i$; this value is $O(n^{2i_{\max}})$. However, the number of spanning trees on n nodes, and thus the size of the search space, is n^{n-2}. Therefore, as Fig. 6.20 illustrates, the proportion of these high-locality individuals is small even for moderate n and diminishes exponentially as n grows. The areas of high locality grow more slowly with increasing problem size n than the overall search space. As a result, we expect GEAs using Prüfer numbers and searching for stars to perform worse with increasing problem size n.

We were able to explain the high locality of Prüfer numbers representing stars by calculating the number of neighbors for different tree structures. The number of neighbors for Prüfer numbers stays constant, whereas for phenotypes the number of neighbors is different for different tree types. For stars, there is a one-to-one correspondence and the number of neighbors is the same for genotypes and phenotypes. All other types of trees like lists or arbitrary trees have a higher number of neighbors than stars (or Prüfer numbers). Furthermore, the areas of high locality are very tiny as they grow with $O(n^{2i_{\max}})$, where $i_{\max} \ll n$, whereas the search space grows with $O(n^{n-2})$. Thus, the locality is in general low and GEAs searching for optimal trees that are different from stars must fail.

Performance

We verify experimentally that GEAs using the Prüfer number encoding do not perform well when searching for good solutions in areas where the locality is low. We present results for GEAs only using one-point crossover and for simulated annealing using only mutation. Both search algorithms are applied to the fully easy one-max tree problem from Sect. 6.1.5.

Simulated annealing (SA) can be modeled as a GEA with population size $N = 1$ and Boltzmann selection (Goldberg 1990a; Mahfoud and Goldberg 1995). In each generation, an offspring is created by applying one mutation step to the parent. Therefore, the new individual has distance 1 to its parent. If the offspring has higher fitness than its parent it replaces the parent. If it has lower fitness it replaces the parent with the metropolis probability $P(T) = \exp(-\frac{f_{offspring} - f_{parent}}{T})$, where f denotes the fitness of an individual. The acceptance probability P depends on the actual temperature T which is reduced during the run according to a cooling schedule. With lowering T, the probability of accepting worse solutions decreases. Because SA uses only mutation, and can in contrast to for example a $(1+1)$ evolution strategy

solve difficult multi-modal problems, we use it as a representative of mutation-based evolutionary search algorithms. For further information about simulated annealing we refer to Cavicchio (1970) and Davis (1987).

In Fig. 6.21, we present results for GEAs with $\mu + \lambda$ selection using one-point crossover and no mutation on 16 and 32 node one-max tree problems. $\mu + \lambda$ selection means that we generate λ offspring from μ parents and that we choose the best μ individuals from all $\mu + \lambda$ individuals as parents for the next generation. This selection scheme assures that a once found best individual is preserved during a GA run and not lost again. The structure of the optimal solution is determined to be either a star, list, or an arbitrary tree. For the 16 node problems, we chose $\mu = \lambda = 400$, and for the 32 node problems $\mu = \lambda = 1500$. We performed 250 runs and each run was stopped after the population was fully converged. Figure 6.22 presents results for using simulated annealing. The start temperature $T_{start} = 100$ is reduced in every step by the factor 0.99. Therefore, $T_{t+1} = 0.99 \times T_t$. Mutation is defined to randomly change one digit of the Prüfer number. We performed 250 runs and each run was stopped after 5000 iterations.

The results in Figs. 6.21 and 6.22 show that if the optimal solution is a randomly chosen star, both search algorithms, the recombination-based GA and the mutation-based SA are able to find the optimal star easily. A search near stars is really a guided search and both algorithms are able to find their way to the optimum. However, if the optimal solution is a list or an arbitrary tree, GEAs can never find the optimal solution and are completely misled. Exploring the neighborhood around an individual in an area of low locality results in a blind and random search. Individuals that are created by mutating one individual, or by recombining two individuals, have nothing in common with their parent(s).

The results show that good solutions can not be found if they lie in areas of low locality. A degradation of the evolutionary search process is unavoidable. Evolutionary search using the Prüfer number encoding could only work properly if the good solutions are stars. Near stars the locality is high and a guided search is possible. Furthermore, the empirical results confirm the theoretical predictions from Sect. 3.3.4 that high locality is a necessary condition for mutation- and recombination-based GEAs. If the locality of an encoding is low, the difficulty of a problem is changed by the representation, and easy problems like the one-max tree problem become difficult and can not be solved any more.

The presented empirical results also shed some light on the contradicting statements about the performance of GEAs using Prüfer numbers. Researchers who investigate problems in which good solution are star-like see acceptable results and favor the use of Prüfer numbers. Other researchers with non-star-like optimal solutions, however, observe low performance and advise not to use the encoding. Furthermore, we have seen that the Prüfer number encoding has low locality. Therefore, Prüfer numbers change the difficulty of the problem. Fully easy problems like the one-max tree problem become more

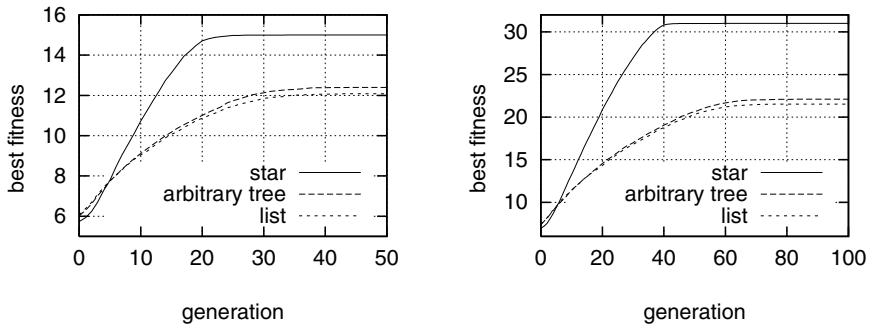


Figure 6.21. The performance of a GA for a 16 (left) and 32 (right) node one-max tree problem. The plots show the fitness of the best individual over the run. The structure of the best solutions has a large influence on the performance of GAs. If the best solution is a star, GAs perform well. If GAs have to find a best solution that is a list or a tree, they degrade and cannot solve the easy one-max problem.

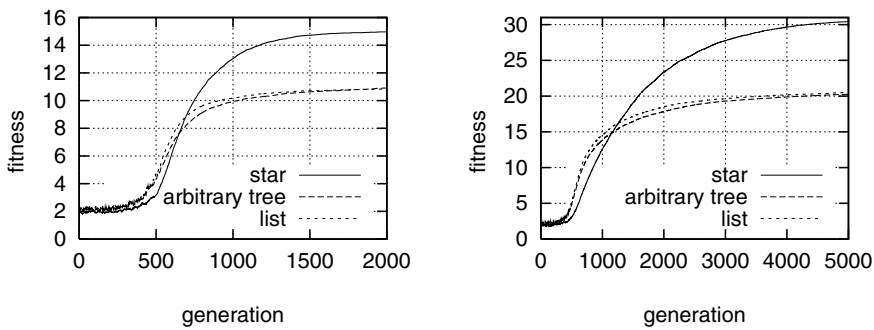


Figure 6.22. The performance of simulated annealing for a 16 (left) and 32 (right) node one-max tree problem. The plots show the fitness of the best individual over the run. As for recombination-based approaches, the mutation-based simulated annealing fails if the best solution is not a star.

difficult, whereas fully difficult problems become more easy. Results about the performance of Prüfer numbers on fully difficult problems are presented later in Sect. 8.1.3.

6.2.5 Summary and Conclusions

This section presented an investigation into the locality of Prüfer numbers and its effect on the performance of GEAs. We started with a historical review on the use of Prüfer numbers. In Sect. 6.2.2, we presented the construction rules for Prüfer numbers. This was followed by a brief overview over the benefits and drawbacks of the encoding. Although the encoding has some remarkable

advantages, it is affected by low locality. Consequently, we focused in the main part of our investigation (Sect. 6.2.4) on the low locality of the Prüfer number encoding. We examined the locality of Prüfer numbers more closely by performing random walks through the search space. This was followed by an analysis of the neighborhood of the genotypes and phenotypes. The analysis showed differences in locality. To explain the differences, we calculated the number of neighbors for Prüfer numbers and trees dependent on the structure of the network. Finally, we empirically verified the theoretical predictions by using recombination-based and mutation-based search algorithms for solving the one-max tree problem.

The historical review showed that there has been a great increase in interest in the Prüfer number encoding over the last few years. However, the suitability of Prüfer numbers for encoding trees is strongly disputed as some researchers report good results whereas others report failure. By performing random walks through the search space, the low locality of the encoding can be nicely illustrated. A small modification of a genotype mostly results in a completely different phenotype. The analysis of the neighborhood of individuals answers the question of whether the locality is low everywhere in the search space, and gives an explanation for the contradicting results from different researchers. The results show that the locality of Prüfer numbers representing stars is high. However, all other types of networks like lists or arbitrary trees lack locality and the genotypic neighbors of a Prüfer number representing a list or an arbitrary tree have on average not much in common with each other. Therefore, the low locality of Prüfer numbers does not reduce GEAs performance in all areas of the solution space to the same extent. This can explain the different results using Prüfer numbers existing in the literature.

To answer the questions of why exactly Prüfer numbers encoding stars have high locality, and how large the areas of high locality are, we investigated the number of neighbors a Prüfer number has. The analysis shows that for Prüfer numbers the number of neighbors remains constant. For phenotypes, however, the number of neighbors varies with the structure of the tree. Stars have as many neighbors as the corresponding Prüfer numbers and therefore, the locality around stars is high. When modifying stars towards lists, the number of phenotypic neighbors increases, which makes it impossible to obtain high locality for problems other than stars. Furthermore, the areas of high locality are only of order $O(n^{const})$, whereas the whole search space grows with $O(n^{n-2})$. Thus, the regions of high locality become very small with increasing problem size n, which reduces the performance of GEAs on larger problems.

The results show that Prüfer numbers have low locality and change the difficulty of problems. Researchers should be careful when using Prüfer numbers on problems of unknown complexity because easy problems become more difficult when using the Prüfer number encoding. As we assume that most of the real-world problems are relatively easy and can be solved using GEAs, GEAs using Prüfer numbers are likely to fail when used on real-world problems.

6.3 The Characteristic Vector Encoding

The characteristic vector (CV) encoding is a common approach for encoding graphs (Davis et al. 1993; Berry et al. 1997; Ko et al. 1997; Dengiz et al. 1997c; Dengiz et al. 1997b; Dengiz et al. 1997a; Berry et al. 1999; Premkumar et al. 2001). Representative examples for the use of the CV encoding for trees can be found in Tang et al. (1997) and Sinclair (1995).

The purpose of this section is to use the framework from Sect. 4 for the analysis of the CV encoding. The investigation shows that CVs are able to represent invalid solutions. Therefore, the encoding is redundant and a repair mechanism is necessary that constructs valid trees from invalid genotypes. Examining the redundancy of CVs shows that they are uniformly redundant but affected by non-synonymous redundancy. Non-synonymous redundancy is a result of the repair process, which brings already extinguished schemata back into the population. As repairing infeasible solutions works like additional mutation and randomizes and prolongs the search process we denote the effect of non-synonymous redundancy as stealth mutation.

In the following subsection, we describe the functionality of the CV encoding. This is followed in Sect. 6.3.2 by a discussion about how to deal with representations that are able to represent invalid solutions. We illustrate that the CV encoding can represent invalid solutions and we propose a repair mechanism for the encoding. In Sect. 6.3.3, we investigate the effects of the repair mechanism. We show that CVs are uniformly redundant because the proposed repair mechanism is unbiased. However, repairing invalid solutions results in non-synonymous redundancy which increases the run duration t_{conv}. The section ends with a brief summary.

6.3.1 Encoding Trees with Characteristic Vectors

We briefly describe the CV encoding and review some of its important properties. The CV encoding can be used for the encoding of trees. Further information and examples for its use can be found in Berry et al. (1994) and Palmer (1994).

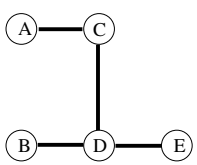


Figure 6.23. A five node tree

A CV is a binary vector that indicates if a link is used or not in a graph. For an n-node graph there exist $n(n-1)/2$ possible links, and a CV of length $l = n(n-1)/2$ is necessary for encoding an n-node graph. All possible links must be numbered, and each link must be assigned to a position in the vector.

In Table 6.3, we give an example of a CV for a 5 node tree. The nodes are labeled from A to E. The link from node A to B is assigned to the first position in the string, the link from A to C is assigned to the second position, and so on. To indicate if the ith link is established, the value at position i is set to one. If no link is established, the value of the allele is set to zero. The tree that is represented by Table 6.3 is shown in Fig. 6.23.

0	1	0	0	0	1	0	1	0	1
A-B	A-C	A-D	A-E	B-C	B-D	B-E	C-D	C-E	D-E

Table 6.3. The CV for the tree in Fig. 6.23

The CV encoding has interesting properties. Firstly, it is able to represent all possible trees. The encoding can also represent non-trees, and we will discuss this problem in the next subsection. Furthermore, all alleles of a CV-encoded genotype have the same contribution to the construction of the phenotype. Therefore, we expect no problems due to non-uniformly scaled alleles, domino convergence, or genetic drift. Finally, the locality d_m of the encoding is high for feasible genotypes. Feasible genotypes that are neighbors correspond to neighboring phenotypes. Two neighboring phenotypes x^p and y^p that have the phenotypic distance $d^p_{x^p,y^p} = 1$ differ in exactly two positions in the feasible genotype, $d^g_{x^g,y^g} = 2$. In general, we can calculate the phenotypic distance d^p between two individuals x and y as

$$d^p_{x,y} = 0.5 d^g_{x,y},$$

where d^g denotes the genotypic Hamming distance and d^p denotes the distance between trees as defined in Sect. 6.1.2. Therefore, the locality of the CV is high if only feasible genotypes are considered. However, as already mentioned, the CV encoding can also represent non-trees. When considering also infeasible solutions, the encoding has low locality and problems with non-synonymous redundancy as the phenotypic neighbors do not correspond to genotypic neighbors any more. We discuss this problem in Sect. 6.3.3.

6.3.2 Repairing Invalid Solutions

We describe how to deal with invalid solutions (non-trees) which can be represented by the CV encoding. The most common approach is to repair invalid genotypes.

Every CV that represents a tree must have exactly $n-1$ ones, the represented graph must be connected, and there are no cycles allowed. This makes the construction of trees from randomly chosen CV demanding as most of the randomly generated CVs are invalid, and not trees. For an n-node network, there are $2^{n(n-1)/2}$ possible CVs, but only n^{n-2} valid trees (Prüfer 1918). The probability of randomly getting a tree is $\frac{n^{n-2}}{2^{n(n-1)/2}} < \frac{2\ln(n)}{\ln(2)n} < 3\ln(n)/n$. Therefore, the chance of randomly creating a CV that represents a tree decreases exponentially with the problem size n (Palmer 1994).

Randomly chosen CVs which should represent a tree can be invalid in two different ways:

- There are cycles in the represented graph.
- The graph is not connected.

We get a cycle for the example in Table 6.3 if we set the first allele (A-B) to one. Then, the genotype does not represent a tree any more because there is a cycle of A-C-D-B-A. Furthermore, if we alter any of the ones in the CV to zero we get two disconnected trees. For example, we could set A-C to zero and get two disconnected subtrees. Subtree one consists of the node A and subtree two consists of a star with center D.

If we do not want to remove invalid solutions from the population, there are two different possibilities to handle invalid solutions: Firstly, we can ignore the invalid solutions and leave them in the population, or secondly, we repair them. Some GEA approaches report to some extent good results when accepting invalid solutions (Orvosh and Davis 1993; Davis et al. 1993). However, when leaving invalid solutions in the population we must ensure that a valid individual is created at the end of the run. Furthermore, we must find a way to evaluate invalid solutions. This can be difficult as for tree problems a fitness function exists for trees, but not for non-trees. Finally, the largest problem is that for tree problems, the probability of generating a valid individual drops exponentially, $O(\exp(-n))$, and therefore, only a very small fraction of the individuals are valid at all. Due to these problems, most of the traditional GEAs choose the second possibility and repair all infeasible solutions.

In general, the repair process for invalid solutions consists of two steps (Berry et al. 1994):

1. Remove links from the tree that cause cycles.
2. Add links to obtain a connected tree.

When repairing a CV that should represent a tree, in a first step, the cycles in the graph must be identified. If we randomly choose $x^g = 1100010100$ (encodes a graph with $n = 5$ nodes), we have a cycle A-C-D-B-A in x^p. Consequently, we remove one randomly chosen link (A-C, C-D, D-B or B-A). When we choose the link A-C we get $y^g = 1000010100$. As there are no more cycles in y^g, we can stop removing links and continue checking whether the graph is fully connected. As there are only three ones in y^g and we have a tree with 5 nodes, the tree can not be connected and we have to add one link. As the node E is separated from the rest of the tree, a link from E to a randomly chosen node A, B, C or D, has to be added. The link C-E is chosen and we finally get $z^g = 1000010110$. A closer look at the repair mechanism shows that the order of the repair steps does not matter.

6.3.3 Bias and Non-Synonymous Redundancy

The CV encoding is redundant as there are infeasible genotypes that have to be repaired to represent a feasible phenotype. Therefore, each phenotype

is encoded by exactly one feasible genotype and a large number of infeasible genotypes. Consequently, we have to investigate whether the representation is uniformly redundant. Otherwise, GEA performance depends on the structure of the optimal solution. Furthermore, we have to investigate whether the encoding is synonymously redundant as synonymous redundancy is necessary for guided search.

A Redundant and Unbiased Encoding

We illustrate that the CV encoding with repair mechanism is a redundant encoding. Furthermore, we show that when using the repair mechanism from the previous subsection the encoding is unbiased that means uniformly redundant.

The genotype-phenotype mapping f_g constructs a valid tree $x^p \in \Phi_p$ with the help of the repair mechanism from every possible feasible or infeasible $x^g \in \Phi_g$. This means that n^{n-2} phenotypes are represented by $2^{n(n-1)/2}$ genotypes. Therefore, the CV encoding is a redundant encoding independently of whether the infeasible individuals are repaired or remain un-repaired in the population. If they remain un-repaired in the population they must be evaluated using the fitness function defined on the phenotypes (that means on trees). Therefore, the infeasible and un-repaired individuals must be assigned in some way to the feasible trees and we have the same situation as when the infeasible genotypes are repaired.

Recognizing that the CV is a redundant encoding, we can use the insights into the effects of redundant encodings from Sect. 3.1. Therefore, we are interested as to whether the CV encoding is biased that means non-uniformly redundant. A closer look at the repair mechanism from Sect. 6.3.2 shows that the removed, respective added links are chosen randomly. We know that an encoding is unbiased if every phenotype is represented on average by the same number of genotypes. The random repair process shows exactly this behavior as it does not favor any particular genotype. Therefore, the CV encoding is unbiased, that means uniformly redundant.

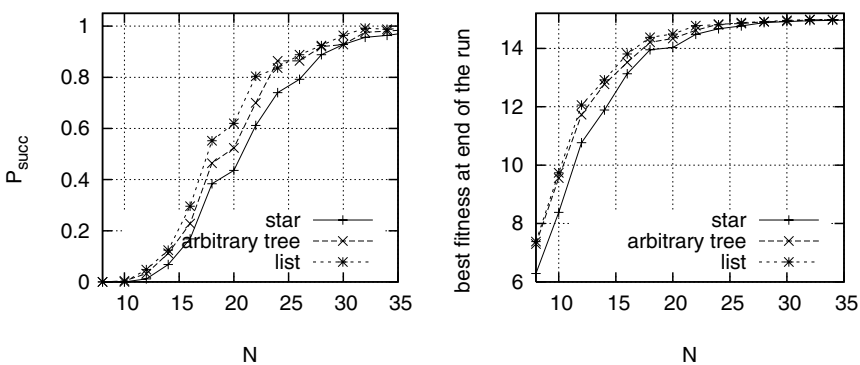
n	$\min(d^p_{rnd,star})$		$d^p_{rnd,MST}$	
	unbiased	CV	unbiased	CV
	μ (σ)	μ (σ)	μ (σ)	μ (σ)
8	3.67 (0.64)	3.66 (0.64)	5.16 (0.99)	5.19 (0.99)
16	10.91 (0.78)	10.91 (0.79)	13.08 (1.07)	13.08 (1.09)
32	26.25 (0.82)	26.23 (0.83)	29.08 (1.31)	29.05 (1.31)

Table 6.4. Minimum distance $\min(d^p_{rnd,star})$ to stars and distance $d^p_{rnd,MST}$ to the MST

To examine whether the CV encoding is unbiased, we randomly create CV-encoded solutions x^g_{rnd}, repair them, and measure for the repaired solutions x^p_{rnd} the average minimum distance $\min(d^p_{rnd,star})$ towards a star and the average distance $d^p_{rnd,MST}$ to the minimum spanning tree (MST). The MST is a spanning tree with minimum costs, where the costs of the tree are the sum

of the distance weights d_{ij} of the used edges (compare (8.3)). In Table 6.4, we present the mean μ and the standard deviation σ of the two distances. We show the average minimum distance to one of the n stars and the average distance to the MST. We randomly created 10,000 CV-encoded individuals and compare the CV encoding to an unbiased representation like the Prüfer number encoding. The nodes are placed randomly on a two-dimensional grid of size 1000×1000 and the distance weights d_{ij} are the Euclidean distances between the nodes i and j. The numbers indicate that the CV is about unbiased this means uniformly redundant.

Because the CV encoding is uniformly redundant, GEA performance should be independent of the structure of the optimal solution. We present in Fig. 6.24 results for the performance of GEAs for a 16 node one-max tree problem (compare Sect. 6.1.5) using only uniform crossover, no mutation, and tournament selection of size 3. The plots show the probability of success $P_{succ} = 1-\alpha$ (finding the optimal solution) and the fitness of the best individual at the end of the run (population is completely converged). We performed 250 runs for T_{opt} is either a star, a list or an arbitrary tree. Before evaluating an infeasible individual in each generation, we repair it according to the algorithm outlined in Sect. 6.3.2, and the repaired CV replaces the infeasible solution. The plots show that GEA performance is about independent of the structure of the optimal solution.



(a) probability of success (b) fitness at end of the run

Figure 6.24. Performance of GEAs using the CV encoding for optimizing a 16 node one-max tree problem, where T_{opt} is either a star, a list, or an arbitrary tree. Invalid solutions are repaired. GEA performance is about independent of T_{opt}. Therefore, the redundant CV encoding using the repair mechanism from Sect. 6.3.2 is unbiased.

Finally, we briefly discuss the possibility of using specific mutation and recombination operators that always create only valid solutions. Then, no repair

steps are necessary and we do not have to worry any more about a bias. Every individual that is created during the GEA's run would be valid. However, one problem is the creation of the initial, feasible solutions as the fraction of feasible solutions is tiny. Furthermore, the creation of thus-like "intelligent" crossover and mutation operators leads to a direct encoding (compare Chap. 7). Then, the genetic operators are, in contrast to the used standard crossover or bit-flipping mutation, not based on the Hamming distance between individuals any more (compare Sect. 3.3.5). Instead, the operators are problem-specific and there exists no explicit representation ($\Phi_g = \Phi_p$).

We have illustrated that the CV encoding is uniformly redundant as phenotypes are represented on average by the same number of infeasible genotypes. Therefore, GEA performance is about independent of the structure of the optimal solution.

Non-Synonymous Redundancy and Stealth Mutation

When recombining two feasible, CV-encoded, parents, the offspring are often underspecified. They do not represent a fully connected tree and the repair mechanism we presented in Sect. 6.3.2 has to insert links randomly to construct a valid solution. Therefore, links which do not exist in both parents could be used for the construction of the offspring. It could also happen that a link that does not not exist in any of the individuals in the population can find its way back into the population by the repair mechanism. This effect caused by the repair process results in cross-competition and should be denoted as *stealth mutation*. Stealth mutation is a result of the non-synonymous redundancy (compare Sect. 3.1.2) of the CV encoding. The CV encoding is non-synonymously redundant as not all infeasible genotypes that encode the same phenotype are similar to each other. We present a brief example. We assume a genotype consisting of only ones. This infeasible genotype is repaired so that all possible phenotypes are constructed with the same probability. Although, on average, all phenotypes are represented by about the same number of infeasible genotypes, it is not possible to determine which infeasible solution encodes which feasible solution and the encoding is non-synonymously redundant. This property of the CV encoding, that the genotype-phenotype mapping of infeasible solutions is random (due to the repair process), results in the situation that not all infeasible genotypes that are decoded to the same phenotype are similar to each other.

When using the notion of BBs, stealth mutation results in a continuous supply of new BBs during a run. Due to cross-competition (compare the effects of non-synonymous redundancy described in Sect. 3.1.2), new BBs are created randomly in the population during the GEA-run, even if they are not present in the start population, or not properly mixed and lost. As the CV encoding is non-synonymously redundant, the difficulty of a problem is changed and easy problems become more difficult to solve as the search process is randomized. GEAs show low performance and the running time of GEAs increases.

In Mühlenbein and Schlierkamp-Voosen (1993) and Thierens and Goldberg (1994), the time until convergence was found for the one-max problem as $t_{conv} = \pi\sqrt{l}/2I$ with the selection intensity I and the string length l. I depends only on the used selection scheme and for tournament selection of size 2 we get $I = 1/\sqrt{\pi}$. With $l = n(n-1)/2$, we get $t_{conv} \approx \pi\sqrt{\frac{\pi}{2}}n$. n denotes the size of the problem (number of nodes).

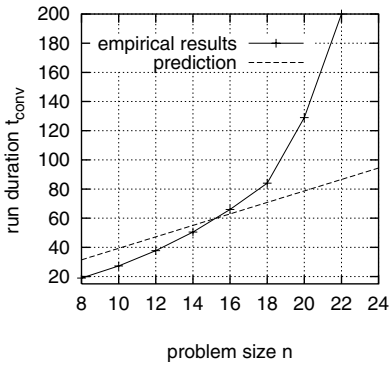


Figure 6.25. Run duration t_{conv} over problem size n for the one-max tree problem using tournament selection of size 2 and uniform crossover. For selectorecombinative GAs, t_{conv} should grow linearly with the problem size n. However, if GEAs use the CV encoding t_{conv} grows due to problems with non-synonymous redundancy approximately exponentially with n.

To illustrate the effects of stealth mutation and non-synonymous redundancy on the performance of GEAs we compare in Fig. 6.25 for the one-max tree problem the theoretical prediction for the run duration t_{conv} with the empirical results for the CV encoding. For the empirical analysis, we use a simple GA with no mutation, tournament selection of size $s = 2$, and uniform crossover. The population size N is large enough to reliably find the optimal solution.

The results show a non-linear dependency of the run duration t_{conv} over the problem size n. The plots indicate that GEAs using CVs struggle because of more repair operations and stealth mutation, which are a result of non-synonymous redundancy. The search for good solutions depends more on the random effects of mutation than on recombination. Due to the non-synonymous redundancy, with higher problem size the probability of randomly finding the correct link decreases, and the run duration of GEAs using CVs increases.

6.3.4 Summary

This section examined important properties of the CV encoding. We started with a description of the encoding and briefly reviewed its important properties. In Sect. 6.3.2, we illustrated the problem of infeasible solutions and how infeasible solutions can be repaired. This was followed in Sect. 6.3.3 by an investigation into the properties of the encoding. We recognized that the CV encoding is redundant and unbiased. Therefore, GEA performance is independent of the structure of the optimal solution. Furthermore, it was discussed

that the repair process results in stealth mutation and non-synonymous redundancy which increases run duration and reduces GEA performance.

We have seen in this section that an encoding which can represent not only valid, but also invalid, solutions encodes a valid phenotype by more than one genotype. Therefore, such an encoding is redundant and the results about redundant encodings from Sect. 3.1 can be used. Redundancy is independent of whether the invalid solutions are repaired, or if they remain untouched in the population. In both situations, it is necessary to evaluate the invalid genotypes and to assign a fitness value to every invalid solution. Furthermore, we have seen that an encoding which can represent invalid solutions is unbiased if the construction of a valid phenotype from an invalid genotype is unbiased and does not favor some phenotypes.

Our investigation in the CV encoding has shown that the encoding is uniformly redundant. Only n^{n-2} valid trees are encoded by n^{n-2} valid and $2^{n(n-1)/2} - n^{n-2}$ invalid solutions. To repair invalid solutions, we presented a randomized and unbiased repair mechanism. The CV encoding is unbiased, and GEAs perform independently of the structure of the optimal solution. However, repairing invalid solutions results in non-synonymous redundancy. We denoted this effect as stealth mutation and showed that it increases the run duration t_{conv}.

6.4 The Link and Node Biased Encoding

The link and node biased (LNB) encoding belongs to the class of weighted encodings and was developed by Palmer (1994). It encodes a tree using weights for the nodes and links of a tree. Additional encoding parameters such as the link-specific and the node-specific bias are necessary to control the importance of link and node weights. The encoding was proposed to overcome the problems of characteristic vectors, predecessor representations (compare also Raidl and Drexel (2000)) and Prüfer numbers. Later, Abuali et al. (1995) compared different representations for probabilistic minimum spanning tree (PMST) problems and in some cases found the best solutions by using the LNB encoding. Raidl and Julstrom (2000) observed solutions superior to those of several other optimization methods using a representation similar to the LNB encoding for the degree-constrained minimum spanning tree (d-MST) problem.

This section examines the LNB encoding using the framework from Chap. 4. It investigates whether the encoding is uniformly redundant, and how the encoding parameters (link-specific and node-specific bias) influence redundancy. The investigation reveals that all versions of the LNB encoding are synonymously redundant. Furthermore, the commonly used simpler version of the encoding, which uses only a node-specific bias, is biased towards stars, and the initial population is dominated by only a few star-like individuals. Focusing on the link-biased (LB) encoding, which only uses a link-specific

bias, reveals that solutions similar to the minimum spanning tree (MST) are overrepresented. The general LNB encoding uses both biases, a node-specific bias and a link-specific bias. Similarly to the node-biased encoding, the LNB encoding is biased towards stars if a node-specific bias is used. When using a small link-bias, the encoding is, in analogy to the LB encoding, biased towards the MST. However, by increasing the link-specific bias, the bias of the LNB encoding is reduced and it becomes uniformly redundant for a large link-specific bias. The LNB encoding has great problems if the two biases are too small, because then only MST-like phenotypes can be represented. At the extreme, if both biases are zero, only the MST can be represented independently of the LNB-encoded genotype.

In Sect. 3.1, we have seen that the performance of GEAs using a synonymously redundant encoding increases if the good solutions are overrepresented, and decreases if they are underrepresented. Therefore, the performance of GEAs using the LNB encoding is high if the optimal solution is similar to stars or MSTs. For all other problems, however, a reduction of GEA performance is unavoidable unless the link-specific bias is high.

In the following subsection, we give a brief description of the LNB encoding. In Sect. 6.4.2, we discuss that the bias of an encoding is equivalent to non-uniform redundancy (compare Sect. 3.1). This is followed by an investigation into whether the NB encoding is uniformly redundant. Section 6.4.4 focuses on the LB encoding and Sect. 6.4.5 presents a population sizing model for the one-max tree problem based on the population model from Sects. 3.1.4 and 6.5.5. In Sect. 6.4.6, we investigate for the LNB encoding the effect of both encoding parameters, the node-specific bias and the link-specific bias. Finally, we present in Sect. 6.4.7 some empirical GEA performance results for the one-max tree problem. The section ends with concluding remarks.

6.4.1 Motivation and Functionality

We review the motivation and the resulting properties of the LNB encoding as described in Palmer (1994), Palmer and Kershenbaum (1994a), and Palmer and Kershenbaum (1994b).

As the costs of a communication or transportation network strongly depend on the length of the links, network structures that prefer short distance links often tend to have higher fitness. Furthermore, it is useful to run more traffic over the nodes near the gravity center of an area than over nodes at the edge of this area (Kershenbaum 1993; Cahn 1998). Thus, it is desirable to be able to characterize nodes as either interior (some traffic only transits), or leaf nodes (all traffic terminates). As a result, the more important a link is, and the more transit traffic that crosses the node, the higher is, on average, the degree of the node. Nodes near the gravity center tend to have a higher degree than nodes at the edge of the network. Hence, the basic idea of the LNB encoding is to encode the importance of a node or link. The more important the node or link is, the more traffic that should transit over it.

When applying this idea to tree encodings, the distance weight matrix, which defines the distance weights between any two nodes, is biased according to the importance of the nodes or links. If a node or link is not important, the modified distance weight matrix should increase the distance weight of all links that are connected to this node. Doing this will result with high probability in a leaf node if the encoded phenotype is constructed from the distance weights by the help of Prim's algorithm.

The Node-Biased Encoding

When using the node-biased (NB) encoding, the genotype b holds weights for each node, and has length n for an n node tree. There is a distance weight matrix which assigns a non-negative distance weight d_{ij} to the $n(n-1)/2$ different edges (ij) of a tree. The values d_{ij} of the distance weight matrix are modified according to b using the weighting function

$$d'_{ij} = d_{ij} + P_2(b_i + b_j)d_{\max}, \tag{6.2}$$

where $b_i \in [0,1]$, $d_{\max} = \max(d_{ij})$, and $i, j \in \{0, \ldots, n-1\}$. The node bias P_2 controls the influence of the node weights b_i on the construction of x^p.

When using the NB encoding for trees, we get the phenotype x^p from the genotype b by calculating the minimum spanning tree (MST) for the modified distance weight matrix D'. Prim's algorithm (Prim 1957) was used in the original work. By running Prim's MST algorithm, nodes i that correspond to a low b_i will probably be interior nodes of high degree in the tree. Nodes j that correspond to high b_j will probably be leaf nodes. Thus, the higher the weights b_i of a node i, the higher is the probability that node i will be a leaf node. To get the tree's fitness, the tree x^p is evaluated by using the original distance weight matrix D.

We illustrate the functionality of the NB encoding with a small example. The NB-encoded genotype $b = \{0.7, 0.5, 0.2, 0.8, 0, 1\}$ holds the node weights. A distance weight matrix for the 5-node problem is defined as

$$D = \begin{pmatrix} - & 2 & 1 & 3 & 4 \\ 2 & - & 5 & 6 & 3 \\ 1 & 5 & - & 4 & 3 \\ 3 & 6 & 4 & - & 10 \\ 4 & 3 & 3 & 10 & - \end{pmatrix}. \tag{6.3}$$

For the construction of the tree x^p from b, we first have to calculate all values d'_{ij} of the modified distance weight matrix. Using $P_2 = 1$, we get for example $d'_{0,1} = 2 + (0.7 + 0.5) \times 10 = 14$. When calculating the remaining $d'_{i,j}$ according (6.2) we get for the modified distance matrix:

$$D' = \begin{pmatrix} - & 14 & 10 & 18 & 12 \\ 14 & - & 12 & 19 & 11 \\ 10 & 12 & - & 14 & 6 \\ 18 & 19 & 14 & - & 19 \\ 12 & 11 & 6 & 19 & - \end{pmatrix}.$$

Using Prim's algorithm for the modified distance matrix D', we finally get the tree illustrated in Fig. 6.26. The represented tree is calculated as the MST using the distance matrix D'. For example, $d'_{0,2} = 10 < d'_{0,i}$, where $i \in \{1, 3, 4\}$. Because the link between node 0 and node 2 has the lowest distance weight d', it is used for the construction of the represented tree.

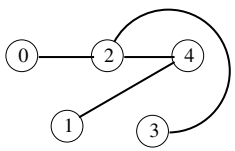


Figure 6.26. An example tree for the node-biased encoding

Palmer noticed in his original work that each b_i modifies a whole row and a whole column in the distance weight matrix. Thus, not all possible solution candidates can be encoded by the NB encoding (Palmer 1994, pp. 66-67).

The Link-Biased Encoding

When using the link-biased (LB) encoding, each genotype holds weights b_{ij} for the $n(n-1)/2$ different edges of a tree. As for the LB encoding, the weights b_{ij} are floating values between zero and one. The original distance weights d_{ij} are modified by the link weights b_{ij} as

$$d'_{ij} = d_{ij} + P_1 b_{ij} d_{\max}, \tag{6.4}$$

where d'_{ij} are the modified distance weights, $d_{\max} = \max(d_{ij})$, and P_1 is the link-specific bias. The parameter P_1 controls the influence of the link-specific weights b_{ij} and has a large impact on the structure of the tree. For $P_1 = 0$, the link-specific weights have no influence and only the MST calculated based on d_{ij} can be represented.

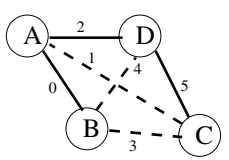


Figure 6.27. An example tree for the LB encoding (The numbers indicate the number of a link)

We illustrate the construction of a tree from the genotype b with a brief example. We use the LB encoding and for representing a tree with $n = 4$ nodes

the genotype is of length $l = n(n-1)/2 = 6$. For the example we want to use the LB-encoded genotype $b_{ij} = \{0.1, 0.6, 0.2, 0.1, 0.9, 0.3\}$. With $P_1 = 1$ and using the distance weights $d_{ij} = \{10, 30, 20, 40, 10, 20\}$ we can calculate the modified distance weights according to (6.4) as $d'_{ij} = \{14, 54, 28, 44, 56, 32\}$. Notice that $d_{\max} = 40$. The represented tree, which is calculated as the MST tree using the modified distance weights d', is shown in Fig. 6.27. The six possible edges are labeled from 0 to 5 and the tree consists of the edges between A and B ($d'_{AB} = 14$), A and D ($d'_{AD} = 28$), and C and D ($d'_{CD} = 32$).

The Link-and-Node-Biased Encoding

As an extension of the NB encoding, Palmer (1994) introduced a second, extended version of the LB encoding with an additional link bias. For the link-and-node-biased (LNB) encoding, the genotype holds weights not only for the n nodes but also for all possible $n(n-1)/2$ links, and has $l = n(n+1)/2$ elements (n node weights b_i and $n(n-1)/2$ link weights b_{ij}). Therefore, the weighting function is

$$d'_{ij} = d_{ij} + P_1 b_{ij} d_{\max} + P_2 (b_i + b_j) d_{\max}, \tag{6.5}$$

with the node weights b_i, the link weights b_{ij}, the link-specific bias P_1, and the node-specific bias P_2. With proper parameter setting, the LNB encoding can represent all possible trees. However, in comparison to the LB encoding, the length of a genotype increases from n to $n(n+1)/2$.

We present a brief example for the LNB encoding. The example genotype holds the node weights $b_i = \{0.7, 0.5, 0.2, 0.8, 0, 1\}$ and the link weights

$$b_{ij} = \begin{pmatrix} - & 0.1 & 0.6 & 0.2 & 0.8 \\ 0.1 & - & 0.1 & 0.9 & 0.5 \\ 0.6 & 0.1 & - & 0.3 & 0.2 \\ 0.2 & 0.9 & 0.3 & - & 0.4 \\ 0.8 & 0.5 & 0.2 & 0.4 & - \end{pmatrix}.$$

With $P_1 = 1$, $P_2 = 1$, and using the distance weight matrix from (6.3) we can calculate $d'_{0,1} = 2 + 0.1 \times 10 + (0.7 + 0.5) \times 10 = 15$. Consequently, we get for the modified distance weight matrix

$$D' = \begin{pmatrix} - & 15 & 16 & 20 & 20 \\ 15 & - & 13 & 28 & 16 \\ 16 & 13 & - & 17 & 8 \\ 20 & 28 & 17 & - & 23 \\ 20 & 16 & 8 & 23 & - \end{pmatrix}.$$

Finally, we calculate the MST using the modified distance matrix D' and get the tree shown in Fig. 6.28.

Using the different variants of the LNB encoding makes it necessary to determine the value of one or two additional encoding parameters, P_1 and

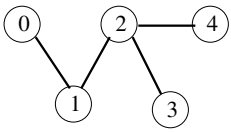


Figure 6.28. An example tree for the LNB encoding

P_2. In the original work from Palmer, only results for the NB encoding with $P_2 = 1$ are presented.

For the different variants of the LNB encoding, the construction of the phenotype can be implemented with a Fibonacci heap and goes with $O(n^2)$. The structure of the represented tree depends not only on the node and link weights, but also on the given distance weights of the links. The same LB, NB, or LNB-encoded genotype can represent different phenotypes if different distance weights are used. Therefore, we assume in our experiments that the distance weights remain constant and do not change during the run of a GA.

6.4.2 Bias and Non-Uniformly Redundant Representations

Representations are redundant, if they assign a discrete, non-infinite number of different phenotypes to genotypes that consist of real values. Each phenotype can be represented by an infinite number of different genotypes. Consequently, the different variants of the LNB encoding are redundant. Furthermore, the representations are synonymously redundant. Genotypes that represent the same phenotype are next to each other in the mutational space. Small mutations of the node weights b_i or link weights b_{ij} often do not change the represented phenotype, or only slightly by one edge. Even large mutations that completely change a node or link weight only result in a change of up to two edges. As a result of the synonymous redundancy of the different variants of the LNB encoding, the models from Sect. 3.1 can be used to predict the influence of the redundant LNB encodings on the performance of GEAs.

In Sect. 3.1, we have seen that uniform redundancy does not change the performance of GEAs in comparison to non-redundant representations. An encoding is uniformly redundant if all phenotypes are represented by the same number of genotypes. However, if some individuals are overrepresented by the encoding, the performance of GEAs is influenced. If the optimal solution is similar to the overrepresented individuals, GEA performance increases. If the optimum is similar to underrepresented individuals, a degradation of GEA performance is unavoidable. As a result, if the encoding is not uniformly redundant, GEA performance depends on the structure of the optimal solution.

We illustrate that a bias of a representation is equivalent to non-uniform redundancy. Therefore, the results about redundant representations from Sect. 3.1 can be also used for biased representations. Palmer defined a biased encoding in his thesis (Palmer 1994, pp. 39) as:

> "It (a representation) should be unbiased in the sense that all trees are equally represented; i.e., all trees should be represented by the

> same number of encodings. This property allows us to effectively select an unbiased starting population for the GA and gives the GA a fair chance of reaching all parts of the solution space."

When comparing this definition of bias to the definition of redundant encodings (compare Sect. 3.1.2), we see that both definitions are essentially the same. An encoding is biased if some individuals are over-, or underrepresented. Furthermore, Palmer correctly recognized, in agreement with the results about redundant encodings, that a widely usable, robust encoding should be unbiased. However, in contrast to Palmer's statement that only unbiased encodings allow an effective search, we have seen that biased (non-uniformly redundant) encodings can be helpful if the encoding is synonymously redundant and overrepresents solutions similar to the optimal solution.

The reader should be careful not to confuse the bias of a representation with the ability to encode all possible phenotypes. Palmer and Kershenbaum (1994a) have already shown that the simple NB encoding is not able to represent all phenotypes. The bias of an encoding describes whether the phenotypes that can be represented are uniformly represented but does not consider the number of represented phenotypes.

As the different variants of the LNB encoding are redundant, the performance of GEAs goes with $O(r/2^{k_r})$. The question arises regarding whether the encodings are non-uniformly redundant, or not. Palmer developed the LNB encoding with the intent to create a uniformly redundant encoding. Therefore, to be able to judge the performance of GEAs using the LNB encoding, we investigate in the following subsections whether the synonymously redundant variants of the LNB encoding are uniformly redundant.

6.4.3 The Node-Biased Encoding

The NB encoding is not capable of representing all possible trees (Palmer 1994). The purpose of this subsection is to investigate whether the LB encoding uniformly encodes the phenotypes that can be encoded. The subsection extends prior work (Gaube 2000; Gaube and Rothlauf 2001) by new results. We start with a distance weight matrix where all d_{ij} have the same value. This is followed by an investigation where the position of the nodes are chosen randomly and the distance weights d_{ij} are the Euclidean distances between the nodes.

All Distance Weights are Equal

We assume that all distance weights d_{ij} are equal, $d_{ij} = d_{kl}$, for $i, j, k, l \in \{0, \ldots, n-1\}$ and $i \neq j, k \neq l$. Thus, different values of d'_{ij} are a result of different node weights b_i (6.2). We denote by $b_l = \min_{i=0}^{n-1} b_i$ the lowest weight of b. It is the weight for the lth node. Then,

$$d'_{il} < d'_{ij} \text{ for } b_l = \min\{b_0, \ldots b_{n-1}\},$$

where $i, j, l \in \{0, \ldots, n-1\}$, $i \neq l$, $i \neq j$ and $l \neq j$. As Prim's algorithm chooses these $n-1$ links with the lowest modified distance weights that do not create a cycle, the only phenotype that could be represented by the NB encoding is a star with center l. Therefore, the LB encoding is uniformly redundant and encodes only stars if all distance weights d_{ij} are the same.

For a tree with n nodes, the number of possible stars is n, whereas the number of all possible trees is n^{n-2}. Thus, only a small fraction of trees could be represented by the node-biased encoding.

non-star	star with center l			
	$l=0$	$l=1$	$l=2$	$l=3$
0%	25.01%	24.97%	24.92%	25.10%

Table 6.5. Average percentage of represented phenotypes ($n=4$)

Although an empirical proof of a theoretical prediction is redundant, Table 6.5 presents an empirical verification of these results for a small 4 node problem. There are 16 possible trees, and 4 of them are stars with center l, where $l \in \{0, 1, 2, 3\}$. For the experiments, we created 1,000 random LB-encoded solutions for 1,000 different distance weights d_{ij}. We see that it is not possible to create non-stars, and that the stars are represented uniformly. As a result, for equal distance weights, the NB representation is uniformly redundant (it represents the n different stars unbiased) but it can only represent a small portion of the solution space (only stars).

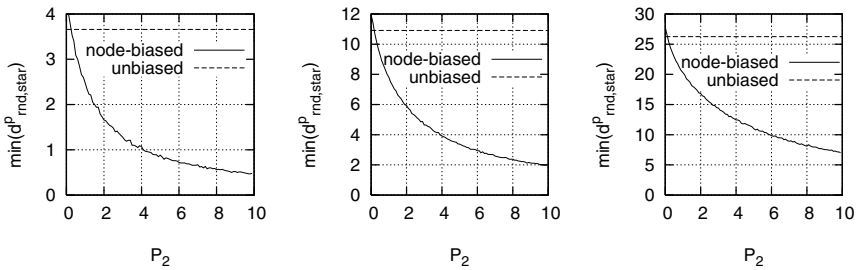
Random Distance Weights

We randomly placed the nodes on a two-dimensional quadratic plane of size 1000×1000 and calculated the distance weights d_{ij} using the Euclidean distance between the nodes i and j (6.1). For the experiments, we randomly created 500 NB-encoded genotypes x^g_{rnd}.

Figure 6.29 shows the average minimum phenotypic distance $\min(d^p_{rnd,star})$ between a randomly created node-biased individual x^p_{rnd} and a star. The distance $d^p_{rnd,star}$ measures how similar the phenotype of a randomly created NB vector is to one of the n stars. If $d^p_{rnd,star}$ is low, x^p_{rnd} has many edges in common with one of the n stars. We performed experiments for 8, 16, and 32 node problems and positioned the nodes 250 times randomly on the 1000×1000 square.

The plots show that in comparison to an unbiased encoding (like Prüfer numbers), where the average distance towards a star stays constant, with increasing node-specific bias P_2, the minimum phenotypic distance $\min(d^p_{rnd,star})$ decreases. Therefore, a randomly created NB-encoded individual becomes more and more star-like and the NB encoding is non-uniformly redundant as it overrepresents solutions similar to stars.

This result is not surprising when we take a closer look at (6.2). The original distance weights d_{ij} are modified by an additional bias. With increasing



(a) 8 node (b) 16 node (c) 32 node

Figure 6.29. Average minimum phenotypic distance $\min(d^p_{rnd,star})$ of a randomly generated NB-encoded genotype to a star. With increasing node bias P_2, the NB encoding is strongly biased towards stars. For large P_2, only one of the n stars can be encoded.

node bias P_2, the influence of d_{ij} relatively decreases and d'_{ij} only depends on the node-specific weights b_i and b_j, and no longer on the distance weights d_{ij}. Therefore, with large P_2, d_{ij} can be completely neglected and the NB encoding can only encode stars. We have the same situation as when all distance weights are the same. Thus, the results described in the previous paragraphs hold true. We see that with increasing P_2, every randomly created NB-encoded individual will decode to a star.

However, not only for very large, but even for reasonable values of P_2, the NB encoding is strongly biased. To investigate how often different phenotypes are represented by a randomly created NB-encoded solution, we ordered the represented phenotypes according to their frequency. In Fig. 6.30, we plot how the cumulative frequencies of the ordered number of copies of a specific phenotype depend on P_2 for a tree with four nodes (there are $4^2 = 16$ different phenotypes). The frequencies are ordered in ascending order. This means that rank 1. corresponds to the phenotype that is most often encoded (encoded with the highest probability), and rank 16. to the phenotype that is encoded with the lowest probability. We performed 1,000 experiments with different node locations and randomly generated 1,000 NB-encoded genotypes for each experiment. The presented values are averaged over these 10^6 different genotypes.

If the encoding is unbiased (uniformly redundant), all individuals are created with the same probability and the cumulative frequency linearly increases with the rank. All 16 possible phenotypes are represented uniformly with probability 1/16=0.0625. However, for the NB encoding, some phenotypes (stars) are created more often. For example, when using the NB encoding with $P_2 = 0.5$, a randomly generated LB-encoded genotype represents with a probability of about 0.5 the same phenotype (rank 1.). The next most frequent phenotype is encoded by a randomly chosen genotype with probability of about 0.25. The line shows that for $P_2 = 0.5$ about 90% of all randomly

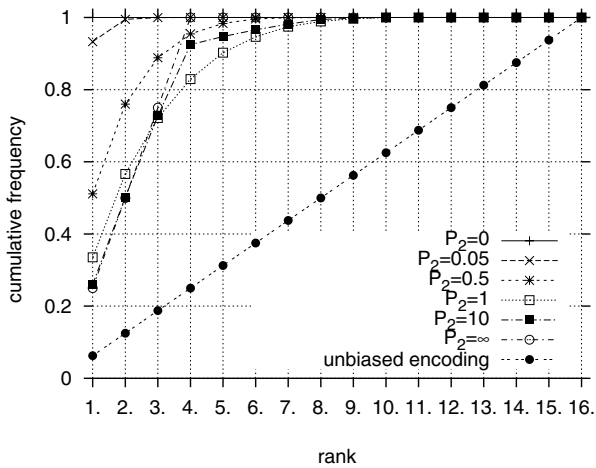


Figure 6.30. Distribution of different phenotypes ($n = 4$)

generated NB-encoded genotypes only represent three different phenotypes. Furthermore, we see that for P_2 large enough ($P_2 > c$), the encoding can only represent a maximum of $n = 4$ different phenotypes which are stars. These 4 stars are encoded unbiased (each of the four individuals is created with 25% probability), similarly to the situation where all distance weights are the same.

The results show that for medium values for P_2 some phenotypes are strongly overrepresented, whereas some others are not represented at all. For lower values of P_2, the node weights b_i have a lower influence on d'_{ij} and the represented phenotypes are MST-like. Figure 6.30 shows that with decreasing P_2, fewer and fewer different phenotypes can be represented. For $P_2 = 0$ only one phenotype, the MST, can be encoded.

Figure 6.31 illustrates the frequency of the first, second, and fifth most frequent phenotype over P_2. The plots are based on Fig. 6.30. We see that with increasing P_2, the diversity decreases, and for $P_2 \gtrsim 10$ only the four most frequent phenotypes (they are the four stars) are uniformly represented with about 25% frequency. With lower P_2, the phenotypes are biased towards the MST, and for $P_2 \to 0$, only one phenotype (the MST) can be represented.

The results show, that the NB encoding is non-uniformly redundant and overrepresents either stars or the MST. For $P_2 \to \infty$, the encoding can only represent stars. When using medium values for P_2, a few star-like individuals dominate a randomly created population, and it is impossible to create some phenotypes. For small values of P_2, the represented phenotypes become more and more MST-like, and for $P_2 \to 0$ only the MST can be encoded.

6.4.4 A Concept for the Analysis of Redundant Representations

In this section, we want to illustrate how the theoretical insights from Sect. 3.1 can be used for predicting the influence of the redundant LB encoding on the performance of GEAs.

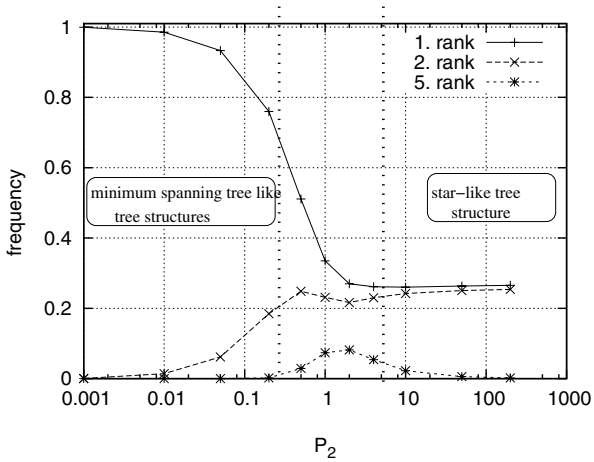


Figure 6.31. Frequency of how often the first, second, and 5th most frequent phenotype is encoded by a randomly chosen LB-encoded genotype ($n = 4$)

Section 6.4.1 illustrated for the LB encoding that the mapping from the genotypes to the phenotypes depends on the link-specific bias P_1. Therefore, to be able to predict the expected GEA performance when using the LB encoding, it must be investigated how P_1 influences the characteristics of the encoding. The following investigation is an example performed for the LB encoding, but the investigation approach is general and can be transferred to any other representation. Factors needing to be examined are:

- Size of the search space.
- Synonymy of the redundant representation.
- Order of redundancy.
- Over- and underrepresentation.

These aspects are discussed in the following paragraphs. When using a (redundant) representation it is important that all possible phenotypes can be represented. A representation should assign at least one genotype to all phenotypes of interest. Otherwise, if no genotype is assigned to some phenotypes, the search space is reduced by the encoding and some possible solutions can never be found by the used search method. The influence of this effect on the performance of a GEA is twofold. If the number of accessible solutions is reduced but the optimal solution is still accessible, GEA performance increases. On the other hand, if the optimal solution is no longer accessible, all search methods must fail. Therefore, a reduction of the phenotypic search space should be avoided if no problem-specific knowledge regarding the optimal solution exists. When using the LB encoding, the number of accessible solutions depends on P_1 (Gaube and Rothlauf 2001). If P_1 is very large, all possible phenotypes can be represented using this encoding. At the other extreme, for P_1 very small ($P_1 \to 0$), only the MST calculated from the distance weights d_{ij} can be represented. As long as $P_1 \gtrsim 1$, every possible phenotype can be encoded as the additional overall bias $P_1 b_{ij} d_{max}$ (compare (6.4)) can

always be larger than d_{ij}. If $P_1 \lesssim 1$ some of the possible trees can not be encoded using the LB encoding.

In our proposed model (Sect. 3.1.4) describing the influence of redundant representations on GEA performance, we assume that non-uniform redundancy changes the initial supply. If we want to use this model for predicting GEA performance we must ensure that the considered representation is synonymously redundant. If a representation is not synonymously redundant, the standard search operators no longer work properly and GEAs fail (compare Section 3.1.2). The LB encoding is synonymously redundant independently of the parameter P_1. Even if the number of accessible solutions decreases with lower values of P_1, a mutation operator always results in the same, or a slightly different, phenotype.

When comparing different types of redundant representations, an important measure is the order k_r of redundancy. In Section 3.1.4, we saw that the population size N goes with $O(2^{k_r})$ for synonymously redundant representations. Therefore, k_r has a strong influence on GEA performance. For the real-valued LB encoding we can assume that k_r remains independent of P_1.

Finally, when using a redundant representation it must be investigated whether some phenotypes are over- or underrepresented. Section 3.1.4 has shown that the necessary population size N goes with $O(1/r)$. In general, the parameter r is problem-specific and depends on the specific instance of a problem. GEA performance remains unchanged if a synonymously redundant representation is uniformly redundant. If a representation is non-uniformly redundant, some instances of a problem will become easier for the search method (those where the optimal solution is overrepresented) and some instances will become more difficult (those where the optimal solution is underrepresented). For the LB encoding, solutions that are similar to the MST are increasingly overrepresented with decreasing P_1. For very small P_1 only a tiny fraction of genotypes represent a solution different from the MST. Only for large values of $P_1 \to \infty$ is the LB encoding uniformly redundant. As a result, there is a continuum between uniform redundancy ($P_1 \to \infty$) and complete non-uniform redundancy ($P_1 = 0$), which can be controlled by the parameter P_1.

In the remaining paragraphs of this subsection, we investigate this continuum and examine how the overrepresentation of specific edges can be controlled by the representation-specific parameter P_1. We want to start with an investigation into how similar a randomly created individual is compared to the MST. The similarity between two trees (the MST and the randomly created individual) is measured by calculating the distance between both trees. The distance between two trees measures the number of different edges (Sect. 6.1.2). In Fig. 6.32, we show the phenotypic distance of a randomly created link-biased individual to the MST for $n = 16$ and $n = 28$. The error bars show the standard deviations. The dotted lines indicate the distance of a randomly created individual towards the MST when using a non-redundant representation (for example Prüfer numbers). The results show that for large values of P_1 a randomly created LB individual has about the same distance

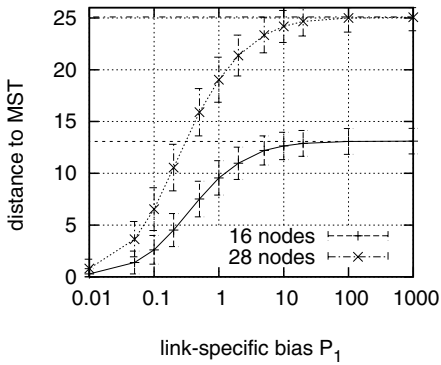


Figure 6.32. Phenotypic distance of a randomly generated LB individual to the minimum spanning tree for trees with $n = 16$ and $n = 26$ nodes. The distance between two individuals indicates the number of links that are different.

towards the MST as a non-redundant encoding. Therefore, it can be assumed that with a large enough P_1, the LB encoding is uniformly redundant. With decreasing values of P_1 the represented trees become more and more MST-like and the LB encoding becomes more and more non-uniformly redundant.

We see that the overrepresentation of a specific solution (the MST) strongly depends on the link-specific bias P_1. To be able to calculate the overrepresentation of specific edges (we need this for the population sizing model we derive in Sect. 6.4.5) we want to examine how P_r, which is the probability that an edge contained in a randomly created LB individual is also contained in the MST, depends on P_1. For non-redundant or uniformly redundant representations the probability P_r^u can be calculated as

$$P_r^u = \frac{2n}{n(n-1)}. \tag{6.6}$$

n	P_r^u	P_r for P_1=1,000,000 $\mu(\sigma)$
8	0.25	0.2497 (0.013)
12	0.1667	0.1665 (0.017)
16	0.125	0.1250 (0.015)
20	0.1	0.0998 (0.010)
24	0.0834	0.0832 (0.012)
28	0.0714	0.0713 (0.012)

Table 6.6. A comparison between P_r^u and P_r for $P_1 = 1,000,000$.

Table 6.6 compares for different problem sizes n, the probability P_r^u for non-redundant representations to empirical results for P_r (mean μ and standard deviation σ) when using a large link-specific bias ($P_1 = 1,000,000$). It can be seen that for large values of P_1 the probability P_r (that an edge contained in a randomly created LB individual is also contained in the MST) equals the probability P_r^u (that a randomly chosen edge is part of the MST). Therefore, for large values of P_1 the encoding becomes uniformly redundant.

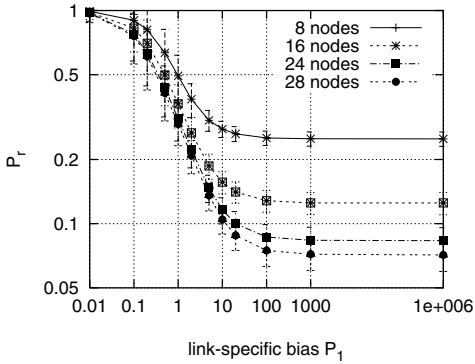


Figure 6.33. Probability that a link of a randomly generated LB-encoded tree is part of the MST over the link-specific bias P_1

Consequently, Fig. 6.33 plots, for the LB encoding, how P_r depends on the link-specific bias P_1. The results show the mean and the standard deviation for 8, 16, 24, and 28 node trees. For large values of P_1 ($P_1 > 100$), P_r equals P_r^u and we get the values shown in Table 6.6. With decreasing P_1, the edges contained in a randomly created individual are also more and more frequently contained in the MST. For small values of P_1, all edges of a randomly created individual are with high probability P_r and also part of the MST.

After discussing how the redundancy of the LB encoding and the overrepresentation of specific edges depends on the link-specific bias P_1, we want to formulate a model based on the results from Sect. 3.1.4, which describes how GEA performance depends on P_1.

6.4.5 Population Sizing for the Link-Biased Encoding

The previous section has shown that with decreasing link-specific bias P_1 the LB encoding overrepresents solutions similar to the MST. This section shows that we are able to give good predictions on how GA performance depends on the link-specific parameter P_1 by combining the population sizing model (6.9) for the network random keys (compare Sect. 6.5.5), which only holds for the uniformly redundant LB encoding (P_1 must be large), with the population sizing model from Sect. 3.1.4, which explains the influence of non-uniformly redundant representations on GEA performance. We formulate the population sizing model for the LB encoding and present experimental results.

In Sect. 6.5.5, we present a population sizing model for the one-max tree problem which was derived for the network random key representation. The network random key representation is almost identical to the LB encoding using large values of P_1. Both encodings are synonymously and uniformly redundant representations defined on real-valued strings of the same length. Only the construction of the tree from the genotypic weights is different. Network random keys use Kruskal's algorithm and do not consider the original distance weights d_{ij}, whereas the LB encoding uses Prim's algorithm for the

construction of the phenotypes and considers d_{ij}. Therefore, the population sizing model for the network random keys is valid for the LB encoding if a large link-specific bias P_1 is used. A large value of P_1 is necessary to ensure uniform redundancy. The model is formulated as (compare (6.9))

$$N = -\frac{\sqrt{\pi}}{4}\ln(\alpha)\sqrt{n(n-1)(n-2)} \approx -\frac{\sqrt{\pi}}{4}\ln(\alpha)n^{1.5},$$

where α is the probability of failure, and n is the number of nodes. It can be seen that the necessary population size N increases with $O(n^{1.5})$. For further information regarding the model, the reader is referred to Sect. 6.5.5.

In the following experiments, the optimal solution for the one-max tree problem is always the MST. We want to calculate for a GA using the LB encoding the population size N that is necessary for finding the optimal solution (the MST) with some probability P_{opt}. The optimal solution is correctly found by a GA if all of the $n-1$ links of the optimal solution are correctly identified. Therefore, $P_{opt} = (1-\alpha)^{n-1}$, where α is the probability of error for one link. We get for

$$\alpha = 1 - \exp\left(\frac{\log(P_{opt})}{n-1}\right).$$

Substituting α into (6.9) results in

$$N = -\frac{\sqrt{\pi}}{4}\ln\left(1-\exp\left(\frac{\log(P_{opt})}{n-1}\right)\right)\sqrt{n(n-1)(n-2)}. \tag{6.7}$$

This population sizing model should give us good predictions for the expected minimal population size using the LB encoding with a large link-specific bias P_1. The large link-bias ensures that the encoding is uniformly biased.

Figure 6.34 shows the theoretical prediction from (6.7) and the experimental results for the LB encoding with $P_1 = 1,000,000$. The plots show the necessary population size N over the problem size n for $P_{opt} = 0.95$. We performed 500 runs for each population size and the resolution for N is 1. We used a standard generational GA with uniform crossover and no mutation. In all runs, we use tournament selection without replacement of size 3 and each run is stopped after the population is fully converged. Because the encoded phenotype depends on the distance weights d_{ij}, for every run we randomly placed the nodes on a 1000x1000 square. d_{ij} is calculated as the Euclidean distance between the two nodes i and j.

Although the population sizing model from (6.7) slightly overestimates the necessary population size N, it still allows a good approximation of the experimental results. As we are mainly interested in investigating the influence of P_1 on the solution quality, and not on the development of a highly accurate population sizing model, we are satisfied with the accuracy of this population sizing model. It can be seen that the necessary population size N increases approximately with $O(n^{1.5})$.

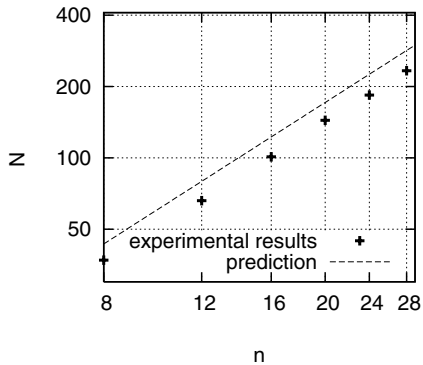


Figure 6.34. Necessary population size N over the problem size n for the one-max tree problem. The optimal solution is the MST and $P_1 = 1,000,000$ to ensure uniform redundancy. The results show that the used population sizing model gives an acceptable approximation of the expected GA performance.

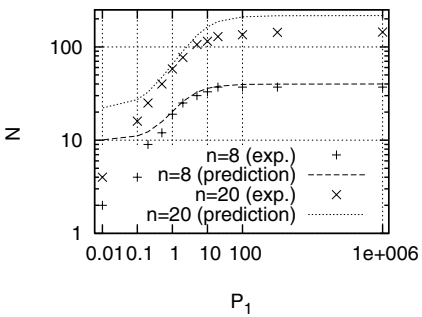
In the following paragraphs, we want to consider that the LB encoding becomes non-uniformly redundant with decreasing P_1. With lower P_1, the links that are contained in the MST are overrepresented by the encoding. Therefore, GEA performance increases and the population size that is necessary to find the optimal solution (the MST) decreases. We know from Sect. 3.1.4 that the necessary population size N goes with $O(2^{k_r}/r)$. r is the number of genotypic BBs that represent the optimal phenotypic BB. For the one-max tree problem we can assume that the size of the BBs k equals one and that each possible link is one phenotypic BB.

We have to determine how the different phenotypic BBs (the possible edges in the tree) are overrepresented by the LB encoding. In Sect. 6.4.4, we have introduced the probability P_r that a link contained in a randomly created individual is also part of the optimal solution. We can assume that the probability P_r is proportional to r ($P_r = const \times r$). Doubling the probability P_r means that a specific link of a randomly created individual is twice as often also contained in the optimal solution (the MST). Therefore, doubling P_r has the same effect as doubling r. Furthermore, we can assume that the character of the LB encoding does not change for different values of P_1 and that k_r remains constant. Therefore, the population size N when using the LB encoding goes with $O(1/P_r)$. From (6.7) we finally get

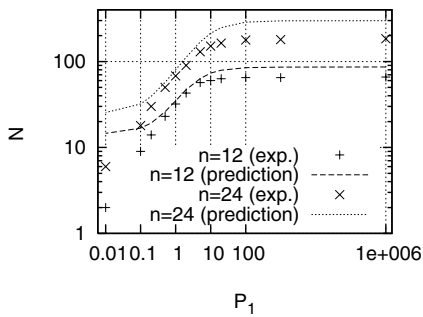
$$N = -\frac{P_r^u}{P_r}\frac{\sqrt{\pi}}{4}\ln\left(1-\exp\left(\frac{\log(P_{opt})}{n-1}\right)\right)\sqrt{n(n-1)(n-2)}, \tag{6.8}$$

where P_r^u indicates P_r for $P_1 \to \infty$ (compare (6.6)). The values of P_r depend on the link-specific bias P_1 and are shown in Fig. 6.33 for different problem sizes n.

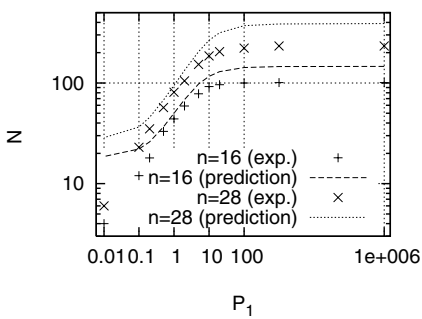
The theoretical predictions and the empirical results for different problem sizes are shown in Figs. 6.35(a) (8 and 20 node one-max tree problem), 6.35(b) (12 and 24 node one-max tree problem), 6.35(c) (16 and 28 node tree one-max tree). The results are split into three plots due to illustrative purposes. The plots show how the necessary population size N depends on the link-specific bias P_1. The probability of finding the optimal solution (the MST) is $P_{opt} = 0.95$. For determining the relationship between P_1 and P_r, which we discussed in Section 6.4.4, we used the results plotted in Fig. 6.33. The lines show the theoretical predictions from (6.8) and the points show the experimental results. In all runs, the optimal solution was the MST and we used the same parameters as for the uniformly redundant LB encoding, whose details are described above.



(a) 8 and 20 node one-max tree



(b) 12 and 24 node one-max tree



(c) 16 and 28 node one-max tree

Figure 6.35. We show how the population size N which is necessary for finding the optimal solution with probability $P_{opt} = 0.95$ depends on the link-specific bias P_1. In all runs, the optimal solution was the MST. The results show that the proposed population sizing model gives good predictions for the expected solution quality. For small values of P_1 the populations size N strongly decreases as the size of the search space collapses and only the optimal solution (the MST) can be represented.

The results show that the proposed population sizing model (6.8) gives us a good prediction on how the performance of a GA depends on the link-specific bias P_1. There is only a small difference between the predicted value for N and the actual experimental results. As expected, the population size N declines with decreasing P_1 and the problem becomes easier to solve for a GA. Furthermore, we can see that for small values of $P_1 < 1$ the necessary population size N strongly declines and the experimental population size drops much faster than predicted. This is because for $P_1 < 1$ (compare Sect. 6.4.4) the LB encoding does not allow us to encode all possible trees and the search space collapses. Only trees that are similar to the MST can be encoded. Small values of P_1 result in high values of P_r (compare Fig. 6.33) which means that most of the links of a randomly created individual are also part of the optimal solution (the MST). In the extreme cases, for $P_1 \to 0$ ($P_r \to 1$), the LB encoding can only encode the optimal solution (the MST) and the necessary population size $N \to 0$.

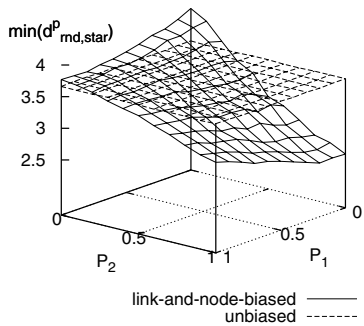
This subsection has illustrated that the proposed theoretical concepts describing the influence of synonymously redundant representations on the performance of GAs can be used for real-valued representations like the LB encoding. The presented results have shown that the proposed theory from Sect. 3.1 predicts the expected GA behavior well.

6.4.6 The Link-and-Node-Biased Encoding

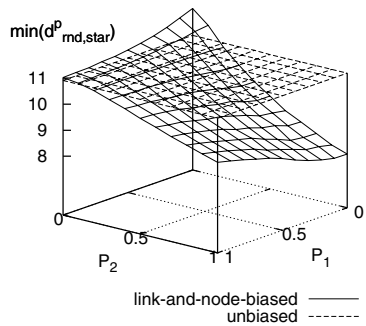
Palmer (1994) proposed the LNB encoding to overcome some of the problems with the node-biased (NB) encoding. In this subsection, we investigate how the non-uniform redundancy which we have noticed for the NB and LB encoding is influenced by the interplay between the two parameters P_1 and P_2.

In analogy to Sect. 6.4.3, we investigate the bias of the LNB encoding by randomly creating LNB-encoded genotypes x^g_{rnd} and measuring their minimal phenotypic distance $\min(d^p_{rnd,star})$ towards one of the n stars. The more links an individual has in common with one of the stars, the more star-like it is and the lower is the distance. In Fig. 6.36, we present results for randomly created LNB-encoded genotypes x^g_{rnd} with 8, 16, and 32 nodes. The average minimum distance $\min(d^p_{rnd,star})$ towards one of the n stars is plotted over P_1 and P_2, and compared to an unbiased encoding (Prüfer numbers). The parameters P_1 and P_2 vary between 0 and 1, and we generated 1,000 LNB-encoded genotypes x^g_{rnd}. The distance weights d_{ij} are the Euclidean distances between the nodes i and j. We present results averaged for randomly positioning the nodes 250 times on a two-dimensional 1000×1000 grid.

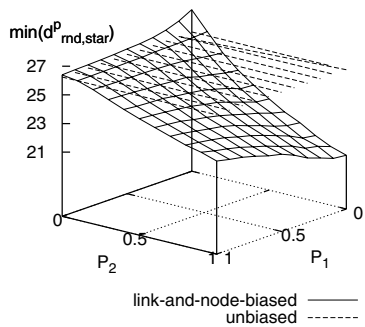
The results show for different n that the non-uniform redundancy of the LNB encoding depends on the node-specific bias P_2. With increasing P_2 the individuals are biased towards stars. With P_2 dominating P_1, we notice the same behavior as for the NB encoding. With increasing P_2, the encoding can only represent star-like structures, and for P_2 large enough, the distance of an individual towards one of the n stars would become zero.



(a) 8 node



(b) 16 node

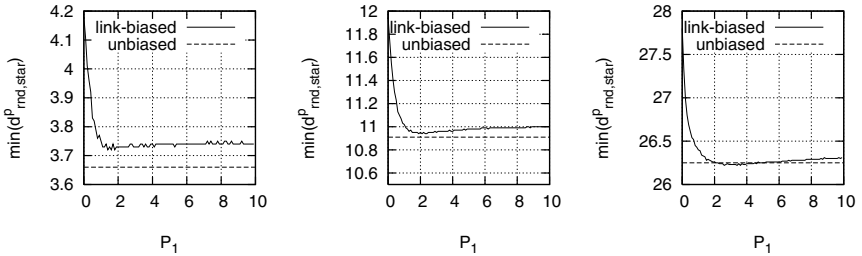


(c) 32 node

Figure 6.36. Average minimum phenotypic distance of a randomly generated LNB-encoded individual with 8, 16, or 32 nodes to a star. By increasing the node-specific bias P_2, an individual is strongly biased towards a star. Higher values for the link-specific bias P_1 result in a lower bias. Small values of P_1 and P_2 result in a bias towards the MST.

As discussed in the previous section, with increasing link-specific bias P_1 the LNB encoding becomes uniformly redundant. To more closely investigate the dependency of the bias on P_1, Fig. 6.37 shows how $\min(d^p_{rnd,star})$ depends on P_1 for $P_2 = 0$. In accordance with the results from Sect. 6.4.4, with increasing P_1, $\min(d^p_{rnd,star})$ stays about constant, and the encoding becomes approximately uniformly redundant. In comparison to non-uniform representations, there is still a small bias. However, as it is very small, we can ignore it.

Finally, we want to emphasize that small values of P_1 and P_2 reduce the performance of the encoding, as the link and node weights have no influence on the construction of the phenotype. The encoding can only represent trees similar to the MST. The results show that with increasing P_2, a randomly created LNB-encoded tree is strongly biased towards stars. By increasing the



(a) 8 node (b) 16 node (c) 32 node

Figure 6.37. Average minimum phenotypic distance of a randomly generated LNB-encoded individual to a star over the link-specific bias P_1. The node-specific bias $P_2 = 0$. For large P_1, the encoding becomes unbiased. For $P_1 \to 0$, only the MST can be encoded.

link-specific bias P_1 the LNB encoding becomes less biased. For P_1 large enough, the encoding is approximately uniformly redundant if the node-bias is small.

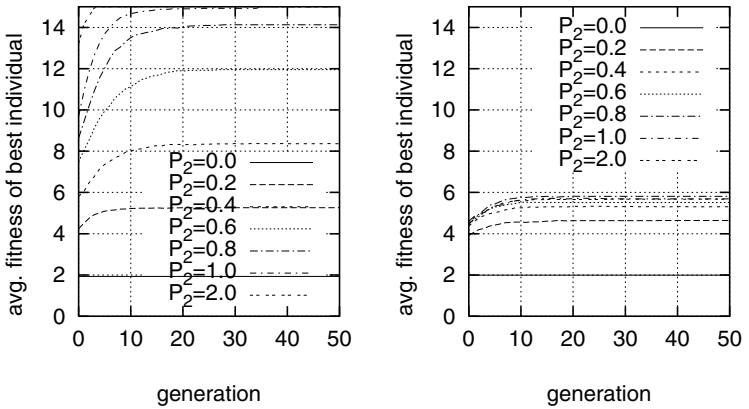
6.4.7 Experimental Results

The previous subsections have shown that the NB encoding as well as the LNB encoding are biased towards stars. With increasing node-bias P_2, star-like structures are strongly overrepresented in a randomly generated population. We know from Sect. 3.1 that redundancy favors genetic search if the optimal solutions are overrepresented by the encoding, and hurts genetic search if the optimal solutions are underrepresented. Therefore, we expect high GEA performance if the optimum is a star, and low GEA performance if the optimum is a non-star such as a random list.

Furthermore, we have seen that with a large link-bias P_1, the encoding becomes uniformly redundant. This means, the performance of GEAs should be independent of the structure of the optimal solution with P_1 large enough. Finally, we know that the LNB encoding can not work for low values of P_1 and P_2 because then the node and link weights have no influence on the structure of the phenotypes and the encoding can only represent MST-like trees. At the extreme, if $P_1 = P_2 = 0$, the genotype has no influence at all, and only the MST can be represented.

To investigate how the performance of GEAs using the LNB encoding depends on P_1 and P_2, we use the one-max tree problem from Sect. 6.1.5. We define the best solution to be either a star or a random list and present the performance of GEAs in Figs. 6.38, 6.39, 6.40, and 6.41. We use a simple GA on a $n = 16$ node problem with only one-point crossover, no mutation, tournament selection of size 3, a population size of $N = 300$, and terminate the run after the population is fully converged. For each parameter setting, we

perform 100 runs with different randomly chosen positions of the 16 nodes (on a two-dimensional 1000×1000 grid). The distance weights d_{ij} are calculated according the Euclidean metric (compare (6.1)).



(a) T_{opt} is a random star (b) T_{opt} is a random list

Figure 6.38. Average fitness of the best individual over the number of generations for different values of the node-specific bias P_2 ($P_1 = 0$). The best solution T_{opt} for the 16 node one-max tree problem is either a star or a list. The results reveal that GAs perform better with increasing P_2 if T_{opt} is a star. If T_{opt} is a list, GAs fail. We see that the performance of GAs using only the LB encoding strongly depends on the structure of the optimal solution.

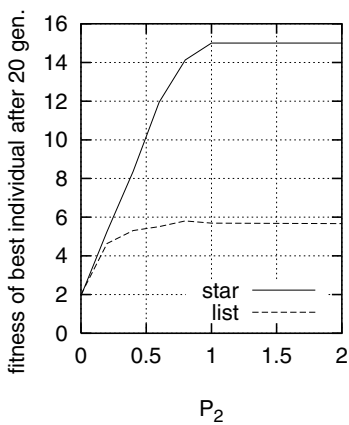
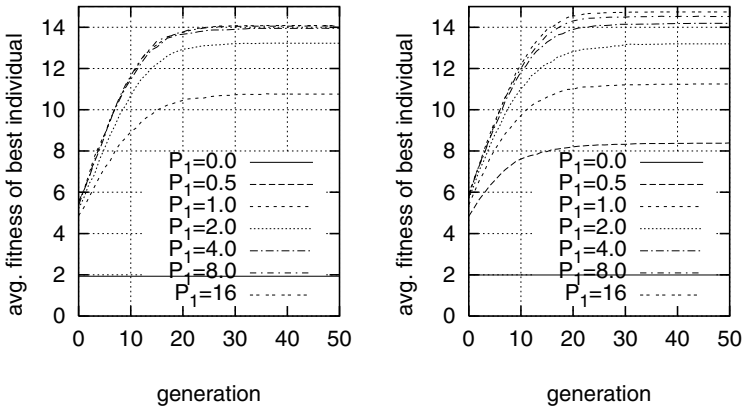


Figure 6.39. We compare for a 16 node one-max tree problem how the fitness of the best individual after 20 generations depends on P_2 for different optimal solutions T_{opt} ($P_1 = 0$). If T_{opt} is a star, the fitness increases with larger P_2. If T_{opt} is a list, GAs fail and the fitness at the end of the run is independent of P_2.



(a) T_{opt} is a random star (b) T_{opt} is a random list

Figure 6.40. Average fitness of the best individual over the number of generations for different values of the link-specific bias P_1 ($P_2 = 0$). The best solution T_{opt} for the 16 node one-max tree problem is either a star or a list. The results reveal that with P_1 large enough, the LB encoding is unbiased, and the performance of GAs is independent of the structure of T_{opt}.

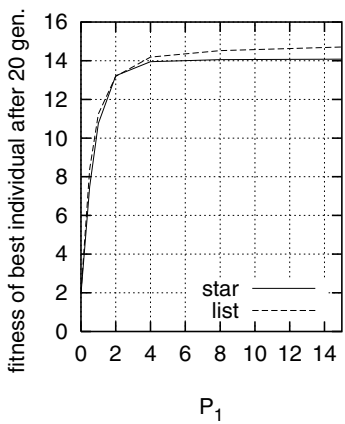


Figure 6.41. We compare for a 16 node one-max tree problem how the fitness of the best individual after 20 generations depends on P_1 for different optimal solutions ($P_2 = 0$). We see that the performance of GAs is independent of the structure of the optimal solution. Therefore, if the link-bias P_1 is large enough, the LB encoding is unbiased.

Figures 6.38(a) and 6.38(b) present the average fitness of the best individual over the run dependent on different values of the node-specific bias P_2 ($P_1 = 0$) if the optimal solution is either a star or a list. We see that with increasing P_2, GEAs find the optimal star much faster, whereas GEAs fail completely when searching for the optimal list. The reader should also notice that with increasing P_2 the initial population becomes more and more star-like, and the average fitness of the best individual in the initial population becomes higher if the optimum is a star. If the node-bias is very small ($P_2 \to 0$) only the MST can be encoded and the only individual a GA can find is the MST. As a result, GAs fail for small values of P_2. The problem with

P_2 becomes more obvious when looking at how the best solution at the end of the run depends on P_2 as illustrated in Fig. 6.39. If the optimum is a star, GEAs perform better with increasing P_2. However, if the optimum solution is a list, GEAs are not able to find the optimal solution.

The situation is different when investigating the influence of the link-specific bias P_1 ($P_2 = 0$) on the performance of GEAs as illustrated in Figs. 6.40 and 6.41. For both problems, GEAs work properly with P_1 large enough. GEAs searching for a star perform as well as when searching for a list. If P_1 is large enough, the encoding is (almost) unbiased and GA performance is independent of the structure of the optimal solution. As we have seen before, the encoding is slightly biased towards lists and GAs perform slightly better when searching for optimal lists. However, as the effect is very small we want to neglect it.

The experimental results confirm the theoretical predictions from the previous subsections. With a large node-specific bias P_2, the LNB encoding is strongly biased towards stars, and GEAs fail if the optimal solution is not a star. With a large link-specific bias P_1, the encoding becomes uniformly redundant and GEAs well perform independently of the structure of the optimal solution. If both biases are very small, only the MST can be represented and GEAs not searching for the MST fail.

6.4.8 Conclusions

Analyzing the notion of biased encodings as given by Palmer (1994) we recognize that a bias is equivalent to non-uniform redundancy. Therefore, we can use the results from Sect. 3.1 for analyzing the performance of GEAs using the LNB encoding. The performance of GEAs using redundant representations goes with $O(r/2^{k_r})$, where r denotes the number of copies that are given to the best phenotypic BB and k_r denotes the order of redundancy. Palmer, who introduced the LNB encoding, drew the conclusion that the

> "... new Link and Node Bias (LNB) encoding was shown to have all the desirable properties ..." (Palmer 1994, pp. 90)

illustrated in Sect. 6.1.6 including those to be unbiased that means uniformly redundant. However, we have seen that this claim is not true. With increasing node-specific bias P_2, the LNB encoding becomes more and more biased towards stars and for $P_2 \to \infty$, the LNB encoding is only able to represent stars. Therefore, GEAs using a large node-specific bias can not work properly.

Fortunately, the encoding becomes uniformly redundant with increasing link-specific bias P_1. For $P_1 \to \infty$, all phenotypes are about uniformly represented, and GEAs perform independently of the structure of the optimal solution.

Finally, the LNB encoding has problems if both biases are small because then the encoding can only represent phenotypes that are similar to the min-

imum spanning tree. At the extreme, for $P_1 \to 0$ and $P_2 \to 0$, the genotype has no influence on the phenotype and only the MST can be encoded.

Because optimal solutions for the optimal communication spanning tree problem (see Sect. 8.2) often tend to be star- or MST-like, the LNB encoding could be a good choice for this problem. In general, however, the encoding has some serious problems, especially when using the simplified NB encoding. Researchers should therefore be careful when using this encoding for other problems because some trees are not encoded at all, and a randomly generated LNB-encoded genotype can be biased towards stars or MSTs.

As a result, we strongly encourage users to use, as long as they have no idea about the structure of the optimal solution, high values for the link-specific bias, and to discard the node-specific bias. Otherwise, GEAs are likely to have large problems in finding optimal non-star or non-MST trees, and a reduction of GEA performance is unavoidable.

6.5 Network Random Keys (NetKeys)

The representation framework we developed in Chaps. 3 and 4 tells us that high-quality representations should be robust, allow genetic operators to work properly, and have high locality. Redundant representations should be synonymously and non-uniformly redundant if no knowledge regarding the optimal solutions exists. However, Prüfer numbers, the CV encoding, and the LNB encoding have shown to have problems with some of these issues. Prüfer numbers have low locality and make easy problems more difficult, the CV encoding is non-synonymously redundant, and the LNB encoding is non-uniformly redundant. Therefore, Prüfer numbers and the CV encoding are not suitable encodings for easy problems as they make these problems more difficult. Only the LNB encoding can result in good GEA performance if the encoding parameters P_1 and P_2 are properly set.

The purpose of this section is to design a new representation by combining the advantageous properties of the CV and the LNB encoding. As a result, we get the synonymously and uniformly redundant *network random key* encoding (NetKeys). The NetKey encoding belongs to the class of weighted encodings. In contrast to other representations such as the CV encoding which can only indicate whether a link is established or not, weighted encodings use weights for the genotype and can thus encode the importance of links. Consequently, an additional construction algorithm is necessary which constructs a valid tree from the genotypic weights (the random key sequence of length $l = n(n-1)/2$).

The section starts by illustrating how the NetKey encoding can be designed by combining the CV encoding with some elements of the LNB encoding. This is followed in Sect. 6.5.2 by the functionality of the NetKeys. We illustrate the random keys which store the importance of the links as weighted vectors, and the construction algorithm which constructs a valid tree from a random key sequence. Section 6.5.3 summarizes the properties of the NetKey encoding.

This is followed by an investigation into whether the synonymously redundant NetKey encoding is uniformly redundant, or not. For randomly created genotypes, we measure the distance towards stars and MST, and provide empirical verification that GEAs using NetKeys perform independently of the structure of the optimal solution. Before closing the section with concluding remarks, Sect. 6.5.5 presents a model for the population sizing and run duration of GEAs using NetKeys for the one-max tree problem.

6.5.1 Motivation

We have seen in Sect. 6.3.3 that the CV encoding is uniformly redundant. A repair mechanism is necessary that assigns all infeasible genotypes to feasible solutions. Because an allele only indicates if a link is established or not, the repair mechanism must rely on random link insertion or deletion. Therefore, the CV encoding is non-synonymously redundant, as not all infeasible genotypes that are assigned to one phenotype are similar to each other. We want to use the functionality of the CV encoding as the basis for NetKeys.

When trying to improve the CV encoding, we have to overcome the problem of non-synonymous redundancy. Furthermore, we must ensure that the new encoding remains uniformly redundant. Therefore, we replace the binary alleles, which only indicate if a link is established or not, by continuous alleles which encode the importance of a link by a weighted value (a randomly chosen number). We have seen in Sect. 6.4 that the link-biased (LB) encoding which uses genotypic weights is synonymously redundant. When combining the link weights of the LB encoding with the principles of the CV encoding, we get the NetKey encoding which is synonymously redundant (all genotypes that represent one phenotype are similar to each other) and non-uniformly redundant (no phenotypes are overrepresented).

By using a weighted instead of a binary encoding, the NetKey encoding inherits most of the properties (for example the synonymous redundancy) from the LB encoding. In contrast to the LB encoding, the alleles of the NetKeys directly encode the importance of a link and not a bias of the distance matrix. Furthermore, we do not use Prim's algorithm (Prim 1957) which constructs the MST from the modified distance matrix, but Kruskal's algorithm (Kruskal 1956). The construction algorithm is unbiased and uses only the genotypic weights and no distance weights for the construction of the phenotype.

6.5.2 Functionality

For describing the functionality of the NetKey encoding, we have to separate the representation into two parts: Firstly, the genotype which is a sequence of random keys. It stores the importance of the links as a weighted vector of length $l = n(n-1)/2$. Secondly, the construction algorithm (genotype-phenotype mapping) which constructs a tree (phenotype) from a random key sequence (genotype).

Random Keys

By substituting the zeros and ones in the CV encoding by continuous values that can describe the importance of the links, the first part of NetKey functionality is defined. However, the idea to use a weight for describing the importance of an allele is not new, and has already been presented in a different context as the so called random key (RK) encoding. For other work about weighted encodings in the context of tree representations the reader is referred to Palmer (1994) or Raidl and Julstrom (2000).

The RK representation for encoding permutations was first presented by Bean (1992). Later, the encoding was also proposed for single and multiple machine scheduling, vehicle routing, resource allocation, quadratic assignment the, and traveling salesperson problems (Bean 1994). Norman and Bean (1994) refined this approach (Norman and Bean 2000) and applied it to multiple machine scheduling problems (Norman and Bean 1997). An overview of using RKs for scheduling problems can be found in Norman (1995). In Norman and Smith (1997) and Norman et al. (1998), RKs were used for facility layout problems. In Knjazew (2000) and Knjazew and Goldberg (2000), a representative of the class of competent GAs (fast messy GA (Goldberg et al. 1993)) was used for solving ordering problems with RKs.

The RK representation uses random numbers for the encoding of a solution. A key sequence of length l is a sequence of l distinct real numbers (keys). The values are initially chosen at random, are floating numbers between zero and one, and are only subsequently modified by mutation and crossover. An example for a key sequence is $r = (0.07, 0.75, 0.56, 0.67)$. Of importance for the interpretation of the key sequence is the position and value of the keys in the sequence. If we assume that $Z_l = \{0, \ldots, l-1\}$ then a permutation σ can be defined as a surjective function $\sigma : Z_l \to Z_l$. For any key sequence $r = r_0, \ldots, r_{l-1}$, the permutation σr of r is defined as the sequence with elements $(\sigma r)_i = r_{\sigma(i)}$. The permutation r^s corresponding to a key sequence r of length l is the permutation σ such that σr is decreasing (for example, $i < j \Rightarrow (\sigma r)_i > (\sigma r)_j$). The ordering corresponding to a key sequence r of length l is the sequence $\sigma(0), \ldots, \sigma(l-1)$, where σ (also denoted as r^s) is the permutation corresponding to r.

This definitions say that the positions of the keys in the key sequence r are ordered according to the values of the keys in descending order. In our example, we have to identify the position of the highest value in the key sequence r ($r_1 = 0.75$). The next highest value is $r_3 = 0.67$. We continue ordering the complete sequence and get the permutation $r^s = 2, 4, 3, 1$. In the context of scheduling problems, this permutation can be interpreted as a list of jobs that are executed on one machine (We start with job 2, then continue with job 4, job 3, and job 1). From a key sequence of length l, we can always construct a permutation of l numbers. Every number between 0 and $l-1$ appears in the permutation only once as the position of each key is unique. Here are some properties of the encoding.

- A valid permutation r^s of l numbers can be created from all possible key sequences r as long as there are no two keys r_i that have the same value ($r_i \neq r_j$ for $i \neq j$ and $i, j \in \{0, l-1\}$). Therefore, every random key sequence r can be interpreted as a permutation r^s.
- There are many possibilities for the construction of a key sequence r from a permutation r^s. All elements r_i can be scaled up by some factor and r still represents exactly the same permutation r^s. As long as the relative ordering of the keys in r remains the same, different key sequences always represent the same permutation. It is necessary that r^s is a permutation of l numbers, otherwise no key sequence r can be constructed from r^s.
- RKs encode both, the relative position of a number in the permutation r^s (encoded by the value of the key at position i in comparison to all other keys) and the absolute position of i in r^s. The relative position of a number i in the permutation r^s is determined by the numbers that precede and follow i. It is determined directly by the weights r_i. All numbers j in the sequence r^s that follow i correspond to lower-valued keys ($r_j < r_i$), whereas all numbers j that precede i correspond to higher-valued keys ($r_j > r_i$). In the context of scheduling problems, all jobs where the corresponding key has a higher value than the ith key are executed before job i, and all jobs with a corresponding key with lower value are executed after i. In contrast, the absolute position of a number i in the permutation r^s cannot be encoded directly, but is only indirectly determined by the value of the ith key. The absolute position describes at which position in the permutation r^s a number i appears. A large value at the ith position results in a position at the beginning of the permutation r^s, and a low value results in a position at the end of r^s.
- The distinction between relative and absolute position of a number in the permutation r^s is important for the synonymity of RKs. The synonymity of a redundant encoding, which is based on the locality of a non-redundant encoding, describes how similar the genotypes are that represent the same phenotype. A representation is synonymously redundant if mutating a genotype changes the corresponding phenotype only slightly. A look at RKs shows that they are synonymously redundant when used for ordering problems. A small change in the genotype (the key sequence r) leads to a small change in the phenotype (the permutation r^s). The change of one key changes the relative position of exactly one number. However, one must be careful with the definition of the phenotypic neighborhood. If the absolute position of the numbers in r^s is relevant for the phenotype, a change of one key is disastrous and the representation is non-synonymously redundant. If the value of the key r_i with the highest value is modified, only the number i changes its relative position in the permutation r^s, but up to l numbers change their absolute position in the permutation. However, as we use RKs to represent a permutation of numbers, only the relative, and not the absolute positions of the numbers in the permutation must be

considered. And for problems where the relative positions of numbers are important, RKs are synonymously redundant.

- When using GEAs with RKs, standard crossover and mutation operators can be used and are expected to work well. No repair mechanism, or problem-specific operators, are necessary when using this encoding for ordering problems. The standard one- or multi-point crossover schemes work well (Bean 1994) because the relative ordering of the positions in the parents is preserved and transferred to the offspring (Fox and McMahon 1991). Due to the synonymous redundancy of the encoding we expect standard mutation operators to work well and to construct offspring that are similar to their parents.

We have seen that RKs have interesting properties. When using them for the encoding of trees, we still have to define exactly how a tree can be constructed from them.

Constructing Trees from Random Keys

After we have presented RKs as the basis for the NetKey encoding, we still have to define a construction algorithm which creates a valid tree from a RK sequence. Both elements, the RKs and the construction algorithm are necessary for the new NetKey encoding. To get a synonymously and uniformly redundant encoding, we demand the construction algorithm to preserve the synonymity of the RK encoding and not to favor some phenotypes but to work uniformly.

We have seen that we are able to give priority to the objects in the permutation when using RKs. As NetKeys use continuous variables that could be interpreted as the importance of the link, it is possible to distinguish between more and less important links. The higher the value of the allele, the higher the probability that the link is used for the tree.

When constructing the tree, the positions of the keys in the key sequence r are interpreted in the same way as for the CV. The positions are labeled and each position represents one possible link in the tree. From a key sequence r of length $l = n(n-1)/2$, a permutation r^s of l numbers can be constructed as illustrated above. Then the tree is constructed from the permutation r^s as follows:

1. Let $i = 0$, T be an empty tree with n nodes, and r^s the permutation of length $l = n(n-1)/2$ that can be constructed from the key sequence r. All possible links of T are numbered from 1 to l.
2. Let j be the number at the ith position of the permutation r^s.
3. If the insertion of the link with number j in T would not create a cycle, then insert the link with number j in T.
4. Stop, if there are $n-1$ links in T.
5. Increment i and continue with step 2.

The construction rule is based on Kruskal's algorithm (Kruskal 1956) and only considers the weights of the RK vector for building the tree. With this rule, we can construct a unique, valid tree from every possible RK sequence. Thus, the NetKey encoding is now completely described: The new encoding uses RKs which allows us to give priority to some links, and the construction rule uses this information and gradually builds a valid tree.

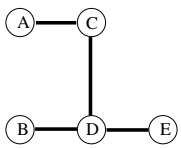


Figure 6.42. A five node tree

We want to illustrate the functionality of the NetKey encoding with an example. We use the key sequence from Table 6.7. The permutation $r^s = 10, 8, 6, 9, 2, 7, 1, 5, 4, 3$ can be constructed from the random key sequence r. We start constructing the tree T by adding the link D-E (position 10) to the tree. This is followed by adding C-D (position 8) and B-D (position 6). If we add the link C-E (position 9) to the tree, the cycle C-E-D-C would be created, so we skip C-E and continue by adding A-C (position 2). Now we have a tree with four edges and terminate the construction algorithm. We have constructed the tree shown in Fig. 6.42.

position	1	2	3	4	5	6	7	8	9	10
value	0.55	0.73	0.09	0.23	0.40	0.82	0.65	0.85	0.75	0.90
link	A-B	A-C	A-D	A-E	B-C	B-D	B-E	C-D	C-E	D-E

Table 6.7. A key sequence r

The computational effort for constructing the phenotype from the genotype is similar for the NetKey and the LB representation. The calculation of the permutation from the key sequence r can be done in $O(l \log(l))$ (sorting an array of l numbers). The process of constructing the graph from the permutation r^s is comparable to repairing an invalid graph that is constructed from a CV and its effort depends on the specific structure of the phenotype.

In analogy to the LB encoding with a large link-specific bias $P_1 \to \infty$, NetKeys are synonymously redundant. A mutation (changing the value of one key) results either in no change of the corresponding phenotype if the relative ordering is not changed, or the change of two edges if the relative position is changed. Therefore, the maximum phenotypic distance $d^p_{x^p,y^p}$ between two neighboring genotypes x^g and y^g is one (compare Sect. 6.1.2 about the definition of distance). The reader should observe that a mutation of one key of the genotype often dramatically changes the absolute positions of the numbers in the permutation r^s. However, the construction rule we defined is

only based on the relative ordering of r^s. Therefore, we do not have to worry about the change of the absolute positions.

6.5.3 Properties

We summarize the properties of the NetKey encoding. The use of the NetKey encoding has some remarkable advantages:

- The encoding is synonymously redundant and standard crossover and mutation operators work properly.
- The encoding allows a distinction between important and unimportant links.
- There are no unfeasible solutions.

In this section, we briefly discuss these properties. In Sect. 3.3.2, we have stipulated that mutation operators must create an offspring which is genotypically and phenotypically similar to its parent. Therefore, a small genotypic distance between two individuals should correspond to a small phenotypic distance. Then, the encoding has high locality. Based on locality, Sect. 3.1.2 introduced the synonymous redundancy of a redundant representation which is equivalent to the high locality of a non-redundant representation. When using synonymously redundant representations, all genotypes that correspond to a phenotype are similar to each other (have small distances). A glance at the NetKey encoding shows that the mutation of one key results either in the same, or in a neighboring tree, which makes it synonymously redundant. Furthermore, in Sect. 3.3.5, we have determined that recombination operators create an offspring which inherits the properties of its parents. In terms of metric, the distance of an individual to its parents should be smaller than the distance between both parents. In terms of links, an offspring should inherit the links from its parents. Standard recombination operators, like n-point or uniform crossover, show this behavior when used for NetKeys: if a link exists in a parent, the value of the corresponding key is high in comparison to the other keys. After recombination, the corresponding key in the offspring has the same high value and is therefore also used with high probability for the construction of the offspring. As a result, both types of operators, mutation and recombination, work well when used for the NetKey encoding.

GEAs using NetKeys are able to distinguish between important and unimportant links. In contrast to the CV encoding, which only stores information about whether a link is established or not, GEAs using NetKeys are able to identify the important links in the tree. As the CV encoding can not store information regarding the importance of a link, the repair process must delete or insert links randomly. High quality links can be accidentally removed, or low quality links can find their way back into the population.

Finally, NetKeys always encode valid trees. No over- or underspecification of a tree is possible. The construction process which builds a tree from a RK

ensures that NetKeys only encode valid solutions. Thus, we do not need an additional repair mechanism.

We see that the NetKey encoding has some remarkable benefits. However, we have not yet investigated whether the encoding is uniformly redundant. We want to do this in the following subsection.

6.5.4 Uniform Redundancy

The NetKey encoding is a synonymously redundant encoding. To ensure that GEAs perform independently of the structure of the optimal solution, NetKeys should be uniformly redundant, i.e. unbiased. This section examines the bias of the NetKey encoding. We measure, in analogy to Sect. 6.3.3, for randomly created NetKey genotypes x^g_{rnd}, the minimum phenotypic distance $\min(d^p_{rnd,star})$ towards stars, and the average phenotypic distance $d^p_{rnd,MST}$ towards the MST. This is followed by empirical evidence of the uniform redundancy of the encoding.

NetKeys are redundant because they encode a finite number of phenotypes using continuous genotypes. We know from Sect. 4.4.1 that GEA performance depends on the location of the optimal solution in the search space if an encoding is non-uniformly redundant. GEAs searching for the optimal solution only perform well if the encoding is not biased towards the low-quality solutions.

Table 6.8. Mean and standard deviation of $\min(d^p_{rnd,star})$ and $d^p_{rnd,MST}$ for randomly created NetKey genotypes x^g_{rnd}

n	$\min(d^p_{rnd,star})$				$d^p_{rnd,MST}$			
	unbiased		NetKey		unbiased		NetKey	
	μ	σ	μ	σ	μ	σ	μ	σ
8	3.67	0.643	3.75	0.602	5.16	0.993	5.24	0.961
16	10.91	0.783	11.00	0.759	13.08	1.072	13.13	1.041
32	26.25	0.818	26.34	0.800	29.08	1.311	29.07	1.319

In Table 6.8, we present for randomly created NetKey genotypes x^g_{rnd} the average minimum distance $\min(d^p_{rnd,star})$ to a star, and the average distance $d^p_{rnd,MST}$ to the MST. The MST (compare (8.3)) is calculated based on the distance weights d_{ij} which are the Euclidean distances between the nodes i and j. The nodes are randomly placed on a two-dimensional 1000×1000 grid. We randomly create 10,000 solutions for each problem instance and show the mean μ and the standard deviation σ of the distances. The numbers indicate that although there is a small bias, we can view the NetKey encoding as uniformly redundant and assume that all phenotypes are represented uniformly. The small bias is expected to be a result of the construction process.

For examining how GA performance depends on the structure of the optimal solution, we use the one-max tree problem from Sect. 6.1.5. Our GA

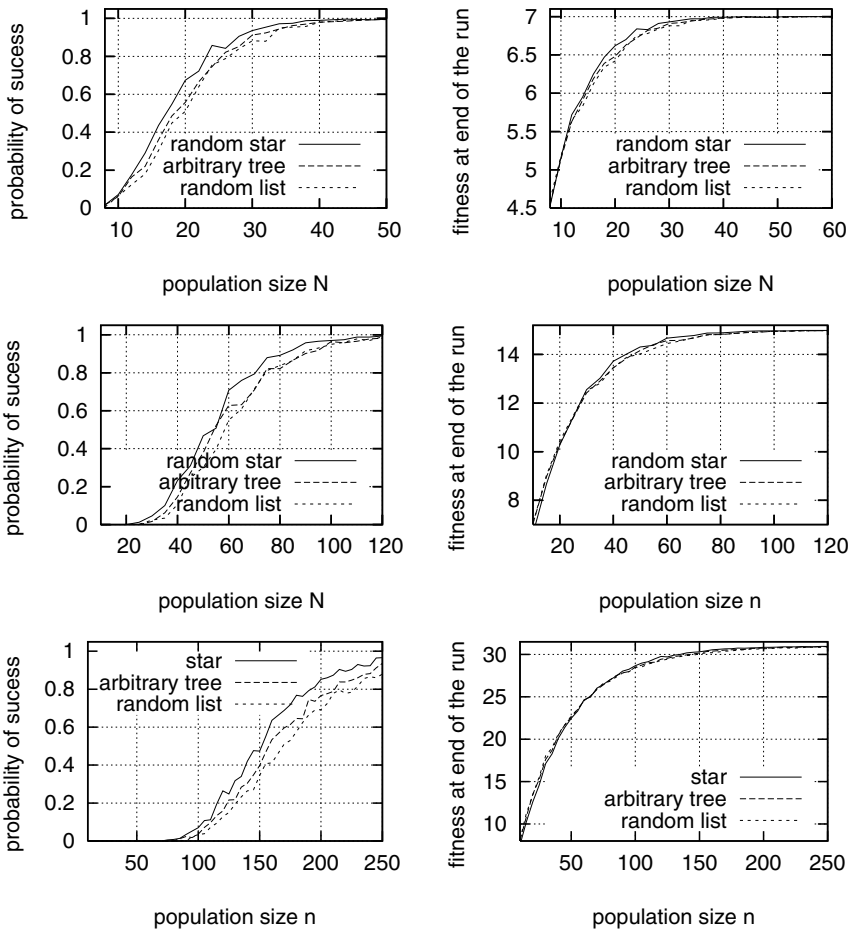


Figure 6.43. Performance of GAs using NetKeys for 8 (top), 16 (middle) and 32 (bottom) one-max tree problems. The plots show either the probability of success (left) or the fitness at the end of a run (right). The GAs search for the optimal star, list, or arbitrary tree. The plots indicate that GA performance is independent of the structure of the optimal solution. Therefore, NetKeys are uniformly redundant.

only uses uniform crossover, no mutation, tournament selection without replacement of size 3, and stops after the population is fully converged. On the left of Fig. 6.43, we show the probability of finding the optimal solution (a randomly chosen star, list, or arbitrary tree) over the population size N. The right figures show the fitness of the best individual at the end of the run over the population size N. Both plots confirm that GEAs perform almost independently of the structure of the optimal solution. The reader should observe that the figures indicate a slightly better performance for stars. We

believe that omitting invalid links during the construction process results in this small influence on GA performance. We ignore this small bias and assume that NetKeys are uniformly redundant. Our results indicate that the NetKey encoding is approximately uniformly redundant and GEAs using NetKeys perform nearly independently of the structure of the optimal solution.

6.5.5 Population Sizing and Run Duration for the One-Max Tree Problem

We examine the necessary population size and run duration of GAs using NetKeys. We present theoretical models for the one-max tree problem (compare Sect. 6.1.5) and provide empirical verification.

Population Sizing

When extending the population sizing equation of Harik et al. (1997) from a binary alphabet to a χ-ary alphabet, we get:

$$N_{min} = -\frac{\chi^k}{2} \ln(\alpha) \frac{\sigma_f}{d} \sqrt{\pi},$$

where χ is the cardinality of the alphabet, α is the probability of failure, σ_f is the overall variance of the function, and d is the signal difference between the best and second best BB. For calculating σ_f we have to investigate how to decide among the competing BBs. For the one-max tree problem we have to find these $n-1$ links the optimal solution is constructed from. The key sequence r that represents a tree with n nodes consists of $l = n(n-1)/2$ different keys. For the construction of the tree, these $n-1$ keys r_i are used that have the highest value. Therefore, we can split the $n(n-1)/2$ different keys r_i in $n/2$ different groups of size $(n-1)$. Finding the optimal solution means that all keys r_i with the links i contained in the optimal solution can be found in one group, which is considered for the construction of the tree and thus contains the keys with the $n-1$ highest values of r_i. A good decision among competing BBs means deciding between the $n/2$ different groups of size $n-1$ and identifying the correct one. A key r_i can belong either to the one group that is considered for the construction of the tree (the key has a high value), or to one of the $n-2$ groups that are not considered. This is similar to the needle in a haystack model and the standard deviation for such a case is (Goldberg et al. 1992)

$$\sigma_f = \frac{\sqrt{2l(n-2)}}{n} = \sqrt{\frac{(n-1)(n-2)}{n}} \approx \sqrt{n}.$$

As we have $n/2$ different partitions, the cardinality χ of the alphabet is $n/2$. Using these results and with $k = 1$ (the one-max tree problem is fully easy,

and there are no interdependencies between the alleles), and $d = 1$, we get an approximation for the population size N_{min}:

$$N_{min} = -\frac{\sqrt{\pi}}{4} \ln(\alpha) \sqrt{n(n-1)(n-2)} \approx -\frac{\sqrt{\pi}}{4} \ln(\alpha) n^{1.5} \tag{6.9}$$

The necessary population size N_{min} goes with $O(n^{1.5})$.

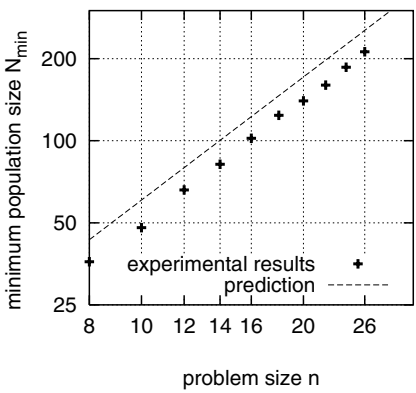


Figure 6.44. Minimum pop size N_{min} for NetKeys over the problem size n for the one-max tree problem. The probability of finding the optimal solution is $P_n = 0.95$. The population size goes with $O(n^{1.5})$.

In Fig. 6.44, the minimum necessary population size N_{min} that is necessary for solving the one-max tree problem with probability $P_n = 1 - \alpha = 0.95$ is shown over the problem size n. We use a simple GA with tournament selection without replacement of size 3, uniform crossover and the NetKey encoding. We perform 500 runs for each population size and for $N > N_{min}$ the GA is able to find the optimum with probability $p = 0.95$ ($\alpha = 0.05$). Although, we have to make some assumptions in our derivation, and the exact influence of the construction algorithm of the phenotype from the genotypic weights is difficult to describe theoretically, the population sizing model gives us a good approximation of the expected population size N which goes with $O(n^{1.5})$.

Run Duration

In Mühlenbein and Schlierkamp-Voosen (1993) and Thierens and Goldberg (1994), the time until convergence is defined as $t_{conv} = \pi\sqrt{l}/2I$ with the selection intensity I and the string length l. I depends only on the used selection scheme and is $I = 3/(2\sqrt{\pi})$ for a tournament size of 3 (Bäck et al. 1997, C 2.3). With $l = n(n-1)/2$, we get $t_{conv} \approx \text{const} \times n$. Therefore, the run duration t_{conv} of a GA should go linearly with the problem size n of the one-max tree problem.

In Fig. 6.45, we show the run duration t_{conv} over the problem size n for tournament selection without replacement of size 3 and uniform crossover. t_{conv} measures the number of generations until the population is completely

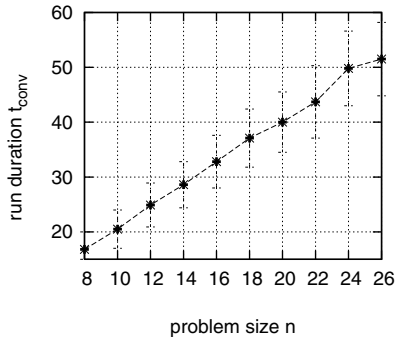


Figure 6.45. Run duration t_{conv} over problem size n for the one-max tree problem using tournament selection without replacement and uniform crossover

converged. The population size was chosen as $N = 2 \times N_{min}$ and N_{min} is from Fig. 6.44. The population size N is large enough to ensure that the optimal solution was found in all 500 runs which we performed for every problem instance. The results show that t_{conv} grows, as predicted, linearly with increasing n.

6.5.6 Conclusions

This section presented the NetKey encoding. We started by illustrating how we can combine characteristic vectors with some elements of the link-biased encoding to get the NetKey encoding. This was followed in Sect. 6.5.2 by the functionality of NetKeys. We explained the principles of random keys and illustrated how we can construct a valid tree from a random key sequence. After all components of the NetKey encoding were defined, we summarized in Sect. 6.5.3 important properties of the new encoding. Because NetKeys are a redundant encoding, Sect. 6.5.4 presented an investigation into the bias of the encoding. Finally, based on existing theory, we developed in Sect. 6.5.5 a population sizing and run duration model for GAs using NetKeys and solving the one-max tree problem.

The section demonstrates that using the framework outlined in Chap. 4 allows theory-guided design of high-quality representations. Based on the insights into the principles of representations, we were able to develop the new NetKey encoding. NetKeys are based on the CV encoding, but use continuous weights for encoding information about the represented tree. Therefore, NetKeys are similar to the link-biased encoding with a large link-specific bias $P_1 \to \infty$, but use a different construction rule for the phenotypes. The investigation into the properties of the NetKeys revealed that the encoding is synonymously and uniformly redundant, that standard crossover and mutation operators work properly, and that the representation does not change problem difficulty.

Based on the presented results, we encourage further study of NetKeys for encoding both trees and other networks. The use of existing theory for

formulating a population sizing model as well as a time to convergence model illustrated the benefits we can get from using existing theory. We encourage users to use existing theory for predicting GEA behavior more frequently. Finally, even though more work is needed, we believe that the properties presented are sufficiently compelling to immediately recommend increased application of the NetKey encoding.

6.6 Conclusions

In this chapter, we used the framework from Chap. 4 for the analysis and design of tree representations. By doing this, we were able to illustrate the relevance of the basic design principles more clearly and to understand the influence of common tree representations on the performance of GEAs.

We started in Sect. 6.1 by providing the necessities for analyzing tree representations. We defined the network design problem and presented the used metric for graphs. Based on the schema analysis for graphs we presented in Sect. 6.1.5 scalable test problems for trees (one-max tree problem and deceptive trap tree problem). The section ended with a review of design criteria for trees as given by Palmer (1994).

Section 6.2 presented an investigation into the properties of the Prüfer number encoding. After an historical review, the construction rule, and known properties of the encoding, we focused in Sect. 6.2.4 on the low locality of the encoding. We performed random walks through the search spaces and showed that the locality of the representation is low. This was followed by an investigation into the locality of neighboring individuals. The section ended with empirical evidence of the theoretical predictions of GEA performance.

Section 6.3 focused on the characteristic vector (CV) encoding as an example for a uniformly redundant representation. We described how trees can be represented by the CV encoding, and how invalid solutions can be handled by repairing them. Examining the repair process for CVs revealed that the encoding is non-synonymously redundant as genotypes that represent the same phenotype are not similar to each other. Therefore, GEA performance is reduced and the time to convergence t_{conv} increases.

In Sect. 6.4, we examined the link and node biased (LNB) encoding. We started by illustrating the motivation for developing the encoding and described its different variants. This was followed by illustrating that the LNB encoding is synonymously and non-uniformly redundant. A closer examination of the non-uniform redundancy revealed that the LNB encoding overrepresents stars if a node-specific bias P_2 is used and it overrepresents the MST if a link-specific bias P_1 is used. Only for $P_1 \to \infty$ does the LNB encoding become uniformly redundant. As a result, GEA performance depends on the structure of the optimal solution. Finally, we verified the theoretical predictions concerning GEA performance by empirical results.

In the last section, we designed the NetKey representation using the insights into the influence of representations on GEA performance. NetKeys are synonymously and uniformly redundant and encode a tree using a weighted vector. We discussed the motivation for designing the new encoding, reviewed its functionality, and summarized important properties of NetKeys. Section 6.5.4 analyzed the bias of the encoding and showed that it is non-uniformly redundant. Furthermore, in Sect. 6.5.5 we used the concepts from Sect. 3.1.4 and developed a population sizing model for NetKeys and the one-max tree problem.

In this section, we used the framework about the influence of representations on GEA performance for the analysis of existing tree representations and for the design of a new representation. The framework allowed us to predict how the performance of GEAs, measured by run duration and solution quality, is affected by the used representation. We were able to compare representations in a theory-based manner, to predict the performance of GEAs using different representations, and to analyze representations guided by theory. The analysis showed that the proposed elements of the framework – redundancy, scaling, and locality – can be used for analyzing representations. We want to briefly summarize the insights our analysis revealed:

Our investigation into the locality of the Prüfer number encoding has shown that the locality is different in different areas of the search space. For trees that are similar to stars, the encoding has high locality, and BB-complexity is the same for the genotypes and phenotypes. However, for non-stars, the encoding has low locality and easy phenotypic problems, where the optimal solution is a non-star, become more difficult to solve when using Prüfer numbers. These insights explain the inconsistent statements about Prüfer number's performance in the literature. If the optimal solution was accidentally star-like, the encoding shows an acceptable performance; if it was non-star-like, GEAs fail.

We presented the CV encoding as an example of a uniformly redundant representation that is non-synonymously redundant. We recognized that an encoding that allows the representation of invalid solutions, like the CV encoding, is redundant. Such an encoding is uniformly redundant if the repair process is unbiased, that means it does not overrepresent some phenotypes. However for the CV encoding, the repair process results in non-synonymous redundancy which has the same effect as low locality for non-redundant representations. Non-synonymous redundancy reduces solution quality and increases the run duration t_{conv}.

The investigation into the synonymously redundant variants of the LNB encoding illustrated the influence of non-uniform redundancy on GEA performance. The LNB encoding overrepresents either stars or the MST. Only for the link-specific bias $P_1 \to \infty$ does the encoding become uniformly redundant. In general, GEAs using the LNB encoding have large problems in finding optimal solutions if these are not a star or the MST. The analysis of the redundant link-biased encoding, which was shown in Sect. 6.4.4, can be generalized and

is helpful for the analysis of redundant representations. By examining the size of the search space, the synonymity of the representation, the order of redundancy, and the over- and underrepresentation of solutions, the influence of a redundant representation on GEA performance can be analyzed.

The section about the NetKey encoding illustrated how the representation framework can be used for the design of high-quality representations. The NetKey encoding is synonymously and uniformly redundant and allows efficient GEA search.

Last but not least, in Sect. 6.1, we presented a schema analysis for graph problems. Using it, we were able to measure the phenotypic problem complexity of a graph problem and to classify problems to be easy or difficult. Furthermore, it can help us to judge if encodings preserve problem difficulty because we can measure if the problem complexity remains constant when mapping the phenotypes on the genotypes. Based on the schema analysis for graphs, we provided a fully difficult deceptive trap and a fully easy one-max tree optimization problem. Both scalable test problems are helpful for comparing the performance of different tree encodings. Furthermore, the test problems allow users to easily examine if GEA performance depends on the structure of the optimal solution. This is important for investigating effects that can be caused by non-uniformly redundant encodings.

This chapter has applied the principles of representations from Chap. 3 to common tree encodings. By identifying Prüfer numbers to have low locality, the LNB encoding to be redundant but biased, and the CV encoding to be uniformly and non-synonymously redundant, we were able to predict the behavior and performance of GEAs using these representations. In general, by applying the presented theory of representations to other not mentioned, or new representations, the behavior of GEAs using these encodings can be much better predicted. Therefore, we want to encourage researchers to use the presented theory for representations from Chaps. 3 and 4 for analyzing other representations.

7

Analysis and Design of Search Operators for Trees

When using GEAs for tree problems it is necessary to encode a solution (tree) such that evolutionary search operators like crossover or mutation can be applied. There are two different possibilities for doing this: *indirect representations* usually encode a tree (phenotype) as a list of strings (genotypes) and apply standard search operators to the genotypes. The phenotype is constructed by an appropriate genotype-phenotype mapping (representation). As seen in the previous chapter, there are many indirect representations for trees such as NetKeys, the LNB encoding, the CV encoding, or Prüfer numbers.

In contrast, *direct representations* encode a tree as a set of edges and apply search operators directly to the set of edges. Therefore, no representation is necessary. Instead, tree-specific search operators must be developed as standard search operators can no longer be used. This chapter uses the insights into representation theory for the analysis and design of search operators for trees. In contrast to Chap. 6, where standard search operators are applied to tree-specific genotypes, here tree-specific search operators are directly applied to the phenotypes as there is no additional genotype-phenotype mapping.

Section 7.1 presents a direct representation for trees (NetDir) as an example for the design of direct tree representations. Search operators are directly applied to trees and problem-specific crossover and mutation operators are developed. The search operators for the NetDir representation are developed based on the notion of schemata from Sect. 6.1.4.

Section 7.2 analyzes the edge-set encoding (Raidl and Julstrom 2003) which encodes trees directly by listing their edges. Search operators for edge-sets are either heuristic considering the weights of edges they include in offspring, or naive, including edges without regard to their weights. Analyzing the properties of the heuristic variants of the search operators shows that solutions similar to the minimum spanning tree are favored. In contrast, the naive variants are unbiased which means that genetic search is independent of the structure of the optimal solution. Although no explicit genotype-phenotype mapping exists for edge-sets and the framework for the design of representations can not be directly applied, it is useful for structuring the analysis of

edge-sets. The results of the analysis show that similarly to non-uniformly redundant representations, edge-sets overrepresent some specific types of trees, and GEA performance increases if optimal solutions are similar to the MST.

Analyzing and developing direct representations nicely illustrates the trade-off between designing either problem-specific representations or problem-specific operators. For efficient GEAs, it is necessary to design either problem-specific representations and to use standard operators such as one-point or uniform crossover, or to develop problem-specific operators and use direct representations.

7.1 NetDir: A Direct Representation for Trees

The purpose of this section is to develop a direct representation for trees (NetDir) and to illustrate that when using direct representations, the design task of finding proper representations is substituted by the search for good crossover and mutation operators. When using GEAs based on the notion of schemata, these problem-specific operators must obey the linkage in the phenotypes and process BBs properly. Therefore, by using direct representations it is not possible to get rid of the difficulties in designing efficient optimization methods.

The section starts with a brief historical review of direct representations for trees. In Sect. 7.1.2, we discuss the properties of direct representations. We demonstrate the benefits and drawbacks of using direct representations for GEAs. Because NetDir directly represents trees as graph structures and not as a list of alleles, standard genetic operators can not be used any more. Therefore, problem-specific operators are necessary. Consequently, in Sect. 7.1.3 we develop mutation and crossover operators for the NetDir representation. The section ends with a short summary.

7.1.1 Historical Review

One of the first approaches to direct representations for trees was presented by Piggott and Suraweera (1993). Offspring individuals are created by randomly copying $n-1$ edges from both parents to the offspring. However, the creation of an offspring does not ensure that the offspring represents a fully connected tree. Therefore, a penalty for invalid solutions is necessary.

Li and Bouchebaba (1999) overcame the problem of invalid solutions and designed more advanced operators such as path crossover and mutation. These operators always generate feasible new solutions. Although Li and Bouchebaba did not compare their new representation with other representations, the results presented were promising.

Raidl (2000) introduced edge crossover and edge insertion mutation for a degree constrained tree problem. New offspring are created by edge crossover in three steps. Firstly, a child inherits all edges which exist in both parents.

Then, the offspring gets the edges which exist only in one parent. Finally, the tree is completed with randomly chosen edges concerning the degree constraints. A direct comparison of this approach to other existing approaches for solving the degree-constrained MST problem is difficult because an additional heuristic for generating good initial solutions was used.

Li (2001) presented an implementation of a direct encoding. The implementation is based on predecessor vectors and the effort for crossover and mutation goes with $O(d)$, where d is the length of a path in a tree. The work illustrates that also a direct representation of trees needs to be represented on a computer system. Based on Raidl (2000), Raidl and Julstrom (2003) presented the edge-set encoding which is analyzed in Sect. 7.2.

7.1.2 Properties of Direct Representations

We have noticed in Sects. 2.1.2 and 2.1.3 that GEAs using direct representations do not use an additional genotype-phenotype mapping $f_g : \Phi_g \rightarrow \Phi_p$. In contrast to the so called indirect representations, where the genotypic space is different from the phenotypic space, $\Phi_g \neq \Phi_p$, the operators are directly applied to the phenotypes $x^p \in \Phi_p$. This situation is illustrated in Fig. 7.1. Therefore, differences between different implementations of direct encodings are not the used representation (there is no genotype-phenotype mapping and all genotypes are trees) but how the genetic operators crossover and mutation are applied to the phenotypes.

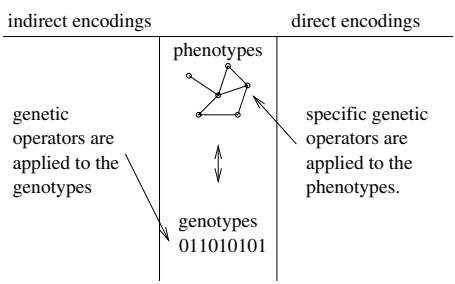


Figure 7.1. Direct versus indirect representations

When using direct representations, it is neither necessary to define a representation (genotype-phenotype) nor genotypes. At a first glance, it seems that the use of direct representations makes life of GEA designer easier as direct representations release us from the pain of designing efficient representations. However, when using direct representations, we are confronted with two other, serious problems:

- Often no standard mutation and recombination operators can be used.
- It is difficult to design proper problem-specific search operators.

We briefly discuss these drawbacks of direct representations. For traditional, indirect representations with standard genotypes, a large variety of different genetic search operators with known properties are available. These standard operators are well examined and well understood. However, when using direct representations, standard operators like n-point or uniform crossover can no longer be used. For each direct representation, problem-specific operators must be developed. Therefore, most of the theory that predicts behavior and performance of GEAs using standard genotypes and standard operators is useless.

Furthermore, the development of proper problem-specific mutation and crossover operators is a difficult task. High quality operators must be able to detect the BBs and propagate them properly. When using direct representations, the design of proper crossover and mutation operators is often demanding as, in general, the phenotypes are not only strings but more complicated structures like for example trees. Furthermore, to use more advanced GEA methods like estimation of distribution algorithms (EDA) or probabilistic model building GAs (PMBGA) become almost impossible. These types of GEAs no longer use standard genetic search operators but build new generations according to a probabilistic model of the parent generations (Mühlenbein and Paaß 1996; Mühlenbein and Mahnig 1999; Harik 1999; Pelikan et al. 1999; Pelikan et al. 1999; Larranaga et al. 1999; Bosman 2003). These search methods are developed for a few standard genotypes (binary or continuous) and result in better performance than traditional simple GAs for decomposable problems. However, because direct representations with non-standard phenotypes and problem-specific genetic operators can hardly be implemented in EDAs or PMBGAs, direct representations can not benefit from these new GEA types.

It is difficult to design high-quality representations when using an indirect representation and standard search operators. However, the task of creating efficient GEAs does not become easier when using direct representations because standard GEA operators can not be used any more and the design of problem-specific operators that can be directly applied to the phenotypes is difficult.

7.1.3 Operators for NetDir

When using direct representations for trees, problem-specific operators must be developed. In the following paragraphs, we present mutation and crossover operators for the NetDir encoding.

Mutation

Section 3.3.2 illustrated that, in general, mutation operators should create offspring which are similar to the parent. Therefore, most mutation operators create offspring with a minimal distance to the parent.

Applying the mutation operator of the NetDir representation to an individual results in a neighboring phenotype. The mutation operator is applied directly to a phenotype $x^p \in \Phi_p$ and results in an offspring $y^p \in \Phi_p$ with phenotypic distance $d^p_{x^p,y^p} = 1$. Therefore, mutation randomly changes one link in the tree. We illustrate the mutation operator in Fig. 7.2. The link $e_{1,5}$ is randomly chosen for deletion. After deleting this link we have two unconnected subtrees. Finally, a node is randomly chosen from each of the two unconnected subtrees and the link connecting the two nodes is inserted ($e_{2,4}$).

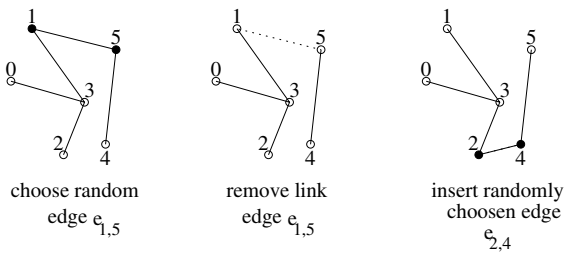


Figure 7.2. The mutation operator for the NetDir representation. The phenotypic distance between parent (left) and offspring (right) is $d^p = 1$.

When using linear genotypes and applying mutation to genotypes, an allele of the string is mutated with mutation probability p_m. Therefore, the probability for an individual to remain unchanged by mutation is $P = (1 - p_m)^l$, where l denotes the length of the string. The situation is different for the NetDir encoding because no linear genotype exists. Mutation for the NetDir encoding is defined as randomly mutating an individual n times with probability p_m, where n is the number of nodes in the graph. Therefore, the probability that the individual remains unchanged is $P = (1 - p_m)^n$.

Crossover

The situation becomes slightly more complicated for the crossover operator. In Sect. 3.3.5, we wanted crossover operators to create offspring that are similar to the parents. The offspring should inherit the high-quality sub-structures of their parents. In terms of metric, crossover operators should ensure that the distances between an offspring and its parents are smaller than the distance between both parents. In terms of schemata, high-quality crossover operators should be able to detect the linkage between the alleles in the string (Harik and Goldberg 1996) and offspring should inherit the high-quality schemata from their parents. Consequently, the crossover operator of the NetDir representation only uses links that exist in the parents for the creation of the offspring. Therefore, the offspring have similar properties than the parents and the schemata are propagated properly.

We denote a complete undirected graph as $G = (V, E)$, where $v \in V$ denotes the n different nodes and $e_{i,j} \in E$ denotes the edge between node $i \in V$ and $j \in V$. Two parents are denoted as $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$.

The two offspring are denoted as $G_{o1} = (V, E_{o1})$ and $G_{o2} = (V, E_{o2})$. The crossover goes with the following scheme:

1. The set of all nodes V is randomly separated into two subsets V_1 and V_2, where $V_1 \cap V_2 = \{\}$ and $V_1 \cup V_2 = V$.
2. All edges $e_{i,j} \in E_1$, where $i, j \in V_1$ are added to G_{o1}. All edges $e_{i,j} \in E_1$, where $i, j \in V_2$ are added to the second offspring G_{o2}.
3. All edges $e_{i,j} \in E_2$, where $i, j \in V_2$ are added to G_{o1}. All edges $e_{i,j} \in E_2$, where $i, j \in V_1$ are added to the second offspring G_{o2}.
4. Do the following steps for each offspring individual separately.
5. There are at least two unconnected subtrees $G_{s1} = (V_{s1}, E_{s1})$ and $G_{s2} = (V_{s2}, E_{s2})$, where $G_{s1} \neq G_{s2}$. Add randomly an edge $e_{i,j} \in (E_1 \cup E_2)$ to the offspring, where either $i \in V_{s1} \wedge j \in V_{s2}$ or $i \in V_{s2} \wedge j \in V_{s1}$.
6. If the offspring is not fully connected, go to 5.

The crossover operator consists of two parts. At first, complete sub-structures are passed from the parent to the offspring (item 1-3). Then, the yet unconnected subtrees are connected by adding links that exist in one of the two parents (item 4-6).

There are several choices for dividing the set of all nodes V into two subsets V_1 and V_2 (item 1). If we assume that the n nodes are numbered, we can use uniform, one-point, or n-point crossover. For uniform crossover the probability that each node belongs to either V_1 or V_2 is 0.5. For one-point crossover we have to choose a crossing point $c \in \{1, 2, \ldots, n-1\}$. The nodes with numbers smaller than c belong to V_1; the nodes with numbers equal or larger than c belong to V_2.

Figure 7.3 illustrates the crossover operator with a small 6-node example. In a first step, the 6 nodes are separated according to uniform crossover into two subsets $V_1 = \{0, 2, 3\}$ and $V_2 = \{1, 4, 5\}$. Then, the edges $e_{0,3}$ and $e_{2,3}$ from parent 1 and the link $e_{1,5}$ from parent 2 are added to offspring 1. Analogously, $e_{1,5}$ and $e_{4,5}$ from parent 1 and $e_{2,3}$ from parent 2 are used for the construction of offspring 2. After copying the subtrees from the parents to the offspring, the remaining separated subtrees must be connected. We do this by randomly copying edges which are able to connect the separated subtrees from the parents to the offspring until the offspring are completely connected. Offspring 1 has three unconnected subtrees $(G_1 = (\{0, 2, 3\}, \{(0, 3), (2, 3)\}), G_2 = (\{1, 5\}, \{(1, 5)\})$, and $G_3 = (\{4\}, \{\}))$. Therefore, the edges $e_{0,1}$, $e_{1,3}$, $e_{3,5}$, $e_{3,4}$, and $e_{4,5}$ can be used for completion of offspring 1. After randomly choosing $e_{3,5}$ and $e_{3,4}$, offspring 1 is fully connected and we can stop. Offspring 2 also has three unconnected subtrees $(G_1 = (\{1, 4, 5\}, \{(1, 5), (4, 5)\}), G_2 = (\{2, 3\}, \{(2, 3)\})$, and $G_3 = (\{0\}, \{\}))$. For offspring 2, the edges $e_{0,1}$, $e_{0,3}$, $e_{1,3}$, $e_{3,4}$, and $e_{3,5}$ can be used for completion. With choosing $e_{0,1}$ and $e_{3,4}$, offspring 2 is fully connected and we can terminate the algorithm.

When measuring the distances between the individuals, the distance between the parents is $d^p_{G_1,G_2} = 3$. The distance of offspring 1 to parent 1 is

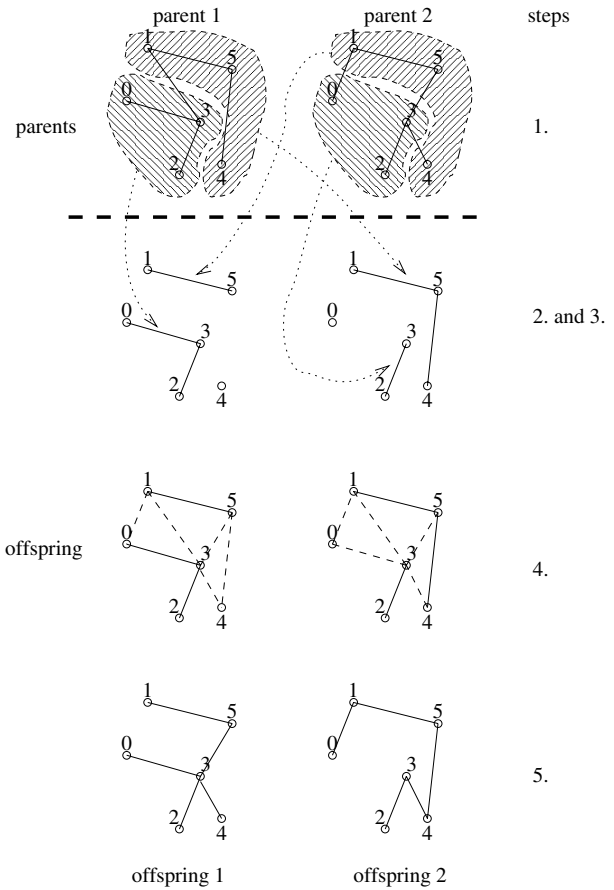


Figure 7.3. The crossover operator of the NetDir encoding. The offspring only inherit information from their parents. No randomly created links are used for the construction of the offspring.

$d^p_{G_1,G_{o1}} = 2$, and to parent 2 is $d^p_{G_2,G_{o1}} = 1$. The distance between offspring 2 and parent 1 is $d^p_{G_1,G_{o2}} = 2$, and to parent 2 is $d^p_{G_2,G_{o2}} = 2$. We see that the distances between the offspring and their parents are smaller or equal to the distances between the parents. The offspring exist mostly of sub-structures of their parents.

7.1.4 Summary

This section presented the direct NetDir representation. After a short historical review of direct tree representations, in Sect. 7.1.2 we discussed the properties of direct representations. Because direct representations directly encode the structure of the problem, standard mutation and crossover operators can often not be used any more. Therefore, we presented in Sect. 7.1.3 tree-specific mutation and crossover operators for the NetDir representation.

The purpose of this section was not just to present another, new representation but to illustrate that the design of efficient GEAs does not become easier when using direct representations. When using direct representations, we need an engineers' intuition and knowledge not for the design of a genotypic search space – it is the same as the phenotypic search space and it is determined a priori by the structure of the problem – but for the design of proper problem-specific genetic operators. Therefore, direct representations do not provide efficient GEAs for free, but in comparison to indirect representations, the overall difficulty of designing efficient GEAs remains the same or even increases.

In this section, we presented the NetDir representation as an example of a direct representation for trees. The NetDir representation directly encodes trees and standard crossover and mutation operators can not be used any more. As most of the existing knowledge about GEA behavior is based on standard operators and standard genotypes, the existing knowledge does not hold any more for direct representations and it is difficult to design high-quality search operators.

7.2 The Edge-Set Encoding

Raidl and Julstrom (2003) proposed another direct representation for trees denoted as the *edge-set encoding*. There are two different variants of the edge-set encoding: heuristic variants where the encoding-specific search operators consider the distance weights of the edges, and non-heuristic variants. Results from applying the edge-set encoding to two sets of degree-constrained MST problem instances indicated that the heuristic variants of the encoding show a higher performance in comparison to other tree encodings such as the Blob code, NetKeys, and weighted encodings (Raidl and Julstrom 2003, p. 238).

This section analyses the bias of the edge-set encoding. A bias of a direct encoding means that the encoding-specific initialization, crossover, and mutation operators prefer a specific type of solution and push a population in this direction. As the heuristic variants of the edge-set encoding prefer edges with low cost, these variants are expected to show a bias towards the MST. In the second part of the section, the performance of edge-sets is investigated for random instances of the optimal communication spanning tree (OCST) problem from Sect. 8.2.1. In contrast to the degree-constraint MST problem used in Raidl and Julstrom (2003), there are no additional constraints regarding the structure of solutions and all possible trees are feasible. As optimal solutions of the OCST problem are biased towards the MST (Rothlauf et al. 2003), heuristic versions of the edge-set encoding are expected to show good performance.

The following subsection summarizes the functionality of the edge-set encoding with and without heuristics. Section 7.2.2 investigates the bias of the

encoding, and Sect. 7.2.3 examines its influence on the performance of evolutionary search for the OCST problem. The section ends with concluding remarks.

7.2.1 Functionality

The edge-set encoding directly represents trees as sets of edges. Therefore, encoding-specific initialization, crossover, and mutation operators are necessary. The following sections summarize the functionality of the different variants with and without heuristics (Raidl and Julstrom 2003).

The Edge-Set Encoding without Heuristics

Initialization

The purpose of the initialization algorithms is to create an unbiased initial solution. Raidl and Julstrom (2003) proposed and investigated three different initialization strategies: PrimRST, RandWalkRST, and KruskalRST. PrimRST overrepresents star-like trees and underrepresents trees similar to lists. RandWalkRST has an average running time of $O(n \log n)$, however, the worst-case running time is unbounded. Therefore, Raidl and Julstrom (2003) recommended the use of the KruskalRST which is based on the algorithm from Kruskal (Kruskal 1956). In contrast to Kruskals' algorithm, KruskalRST chooses edges $e_{i,j}$ not according to their corresponding distance weights d_{ij} but randomly. KruskalRST has a small bias towards star-like trees (which is lower than the bias of PrimRST).

procedure KruskalRST(V, E):
$T \leftarrow \emptyset,\ A \leftarrow E$; //$E$ is the set of available edges $e_{i,j}$
while $|T| < |V| - 1$ **do**
  choose an edge $\{(uv)\} \in A$ at random;
  $A \leftarrow A - \{(e_{u,v})\}$;
  if u and v are not yet connected in T **then**
    $T \leftarrow T \cup \{(e_{u,v})\}$;
return T.

A spanning tree T of an undirected graph $G(V, E)$ with the set E of edges and the set V of nodes is a subgraph that connects all vertices of G and contains no cycles.

Recombination

To obtain an offspring T_{off} from two parental trees T_1 and T_2 with the edge sets E_1 and E_2, KruskalRST is applied to the graph $G_{cr} = (V, E_1 \cup E_2)$. Instead of KruskalRST, in principle PrimRST and RandWalkRST can be also used. The crossover operator has high heritability as in the absence of

constraints, only parental edges are used to create the offspring. Crossover becomes more complicated for constraint MST problems as it is possible that the RST algorithm can create no feasible tree from $G_{cr} = (V, E_1 \cup E_2)$. Then, additional edges have to be chosen randomly to complete an offspring.

Raidl and Julstrom (2003) distinguished two different recombination operators: the variant previously described is denoted KruskalRST crossover. The second variant is denoted KruskalRST* crossover. When using this variant, in a first step all edges $(E_1 \cap E_2)$ are included in the offspring T_{off}. Then T_{off} is completed by applying KruskalRST to the remaining edges $(E_1 \cup E_2) \backslash (E_1 \cap E_2)$. Results from Raidl and Julstrom (2003) indicate a better performance of KruskalRST* for the degree-constraint MST problem.

Mutation

The mutation operator randomly replaces one edge in the spanning tree. This replacement can be realized in two different ways. The first variant of the mutation operator randomly chooses one edge that is not present in T and includes it in T. Then, one edge from the cycle is randomly chosen and removed ("insertion before deletion"). The second variant first randomly deletes one edge from T and then connects the two disjoint connected components using a random edge not present in T ("deletion before insertion"). The running time is $O(n)$ if there are no additional constraints.

The Edge-Set Encoding with Heuristics

The following paragraphs describe how heuristics that rely on the distance weights d_{ij} can be included in the edge-set encoding. Raidl and Julstrom (2003) introduced these variants of the edge-set encoding due to the assumption that in weighted tree optimization problems optimal solutions often prefer edges with low distance weights d_{ij}.

Heuristic Initialization

To favor low-weighted edges when generating the initial population, the algorithm KruskalRST starts by sorting all edges in the underlying graph according to their distance weights d_{ij} in ascending order. The first spanning tree is created by choosing the first edges in the ordered list. As these are the edges with lowest distance weights, the first generated spanning tree is a MST. Then, the k_d edges with lowest distance weights are permuted randomly and another spanning tree is created using the first edges in the list. The heuristic initialization results in a strong bias towards the MST. With increasing k_d, the bias of randomly created trees towards the MST is reduced. The number of edges which are permuted increases according to

$$k_d = \alpha(i-1)n/N,$$

where N denotes the population size, i is the number of the tree that is actually generated ($i = 1, \ldots, N$) and α, with $0 \leq \alpha \leq (n-1)/2$, is a parameter that controls the strength of the heuristic bias.

Heuristic Recombination

The heuristic recombination operator is a modified version of KruskalRST* crossover. Firstly, the operator transfers all edges $E_1 \cap E_2$ that exist in both parents T_1 and T_2 to the offspring. Then, the remaining edges are chosen randomly from $E' = (E_1 \cup E_2) \backslash (E_1 \cap E_2)$ using a tournament with replacement of size two. This means, the distance weights d_{ij} of two randomly chosen edges are compared and the edge with the lower distance weight is inserted into the offspring (if no cycle is created). If the underlying optimization problem is constrained, it is possible that the offspring has to be completed using edges not in E'.

Heuristic Mutation

The heuristic mutation operator is based on mutation by "insertion before deletion". In a pre-processing step, all edges in the underlying graph are sorted according to their weights in ascending order. Doing this, a rank is assigned to every edge. The rank one is assigned to the edge with the lowest weight. To favor low-weighted edges, the edge that is inserted by the heuristic mutation operator is not chosen randomly but according to its rank

$$R = \lfloor |\mathcal{N}(0, \beta n)| \rfloor \bmod m + 1,$$

where $\mathcal{N}(0, \beta n)$ is the normal distribution with mean zero and standard deviation βn and $m = n(n-1)/2$. β is a parameter that controls the bias towards low-weighted edges. If a chosen edge already exists in T, the edge is discarded and the selection is repeated.

7.2.2 Bias

As we have seen in Sect. 3.1 a redundant representation is unbiased if all possible phenotypes are represented by, on average, the same number of genotypes (compare also Sect. 6.3.3). Consequently, a search operator is unbiased if it does not overrepresent specific solutions, and the application of the search operator alone does not modify the statistical properties of a population. An unbiased search operator allows a uniform, non-directed search through the search space. A biased representation or operator should only be used if it is known a priori that the optimal solution of the underlying optimization problem is similar to the overrepresented solutions (compare Sect. 3.1.4). In contrast, unbiased representations or operators should be used if no a priori problem-specific knowledge is available. Then, the probability of finding the optimal solution is independent of the structure of the optimal solution.

The following paragraphs investigate the bias of the edge-set encoding for randomly created trees with $n = 10$ and $n = 16$ nodes. To every edge $e_{i,j}$ a non-negative distance weight d_{ij} is associated. Two possibilities for choosing the distance weights d_{ij} are considered:

- **Random weights:** The real-valued weights d_{ij} are generated randomly and are uniformly distributed in $]0, 100]$.
- **Euclidean weights:** The nodes are randomly placed on a 1000×1000 grid. The distance weights d_{ij} between the nodes i and j are the Euclidean distances between the two nodes.

As the distance weights d_{ij} are randomly created and $d_{ij} \neq d_{kl},\ \forall i \neq l, j \neq l$, we can assume that there is an unique minimum spanning tree (MST) for every problem instance. T is the MST if $c(T) \leq c(T')$ for all other spanning trees T', where $c(T) = \sum_{e_{i,j} \in T} d_{ij}$. The similarity between two spanning trees T_i and T_j can be measured using the distance $d^p_{T_i,T_j} \in \{0, 1, \ldots, n-1\}$ (compare Sect. 6.1.2) as $d^p_{T_i,T_j} = \frac{1}{2} \sum_{u,v \in V,\ u<v} |l^i_{uv} - l^j_{uv}|$, where l^i_{uv} is 1 if $e_{u,v}$ exists in T_i and 0 if it does not exist in T_i.

Initialization

Raidl and Julstrom (2003) examined the bias of different initialization methods and found KruskalRST to be slightly biased towards stars. As the bias is sufficiently small and due to its lower running time it is preferred in comparison to RandWalkRST and PrimRST, which show a stronger bias towards stars.

Table 7.1 shows the average distances $d^p_{rnd,MST}$ between the MST and randomly generated trees (the standard deviations are shown in brackets). For each problem instance (1000 of each type) we generated 10,000 random solutions using either an unbiased encoding (Prüfer numbers), KruskalRST (p. 225), or the heuristic initialization (p. 226). For the heuristic initialization, α was set either to $\alpha = 1.5$ as recommended in Raidl and Julstrom (2003) or to the maximum value $\alpha = (n-1)/2$, which results in the lowest bias. The results confirm that for KruskalRST no bias towards the MST can be observed. Furthermore, the heuristic versions show a strong bias towards the MST even when using a large value of α.

Recombination

To investigate whether the crossover operator of the edge-set encoding leads to an overrepresentation of MST-like individuals, we randomly generate an initial population of 500 individuals and apply only the crossover operator iteratively. As no selection operator is used, no selection pressure pushes the population to high-quality solutions. The crossover operator is unbiased if the statistical properties of the population do not change by applying crossover alone. In our

Table 7.1. Distances $d^p_{rnd,MST}$ between random trees and MST

tree size	$n = 10$		$n = 16$	
weights	Euclidean	random	Euclidean	random
unbiased	7.20 (1.1)		13.12 (1.2)	
KruskalRST	7.20 (1.1)	7.20 (1.1)	13.13 (1.2)	13.13 (1.2)
heuristic ($\alpha = 1.5$)	1.06 (1.2)	0.84 (1.1)	1.87 (1.8)	1.39 (1.7)
heuristic ($\alpha = (n-1)/2$)	3.92 (2.7)	3.85 (2.7)	8.82 (4.4)	8.87 (4.6)

experiments we measure in each generation the average distance $d_{mst-pop} = 1/N \sum_{i=0}^{N-1} d^p_{T_i,MST}$ of the individuals T_i in the population towards the MST. If $d_{mst-pop}$ decreases, the crossover operator is biased towards the MST. If $d_{mst-pop}$ remains constant, the crossover operator is unbiased regarding the MST.

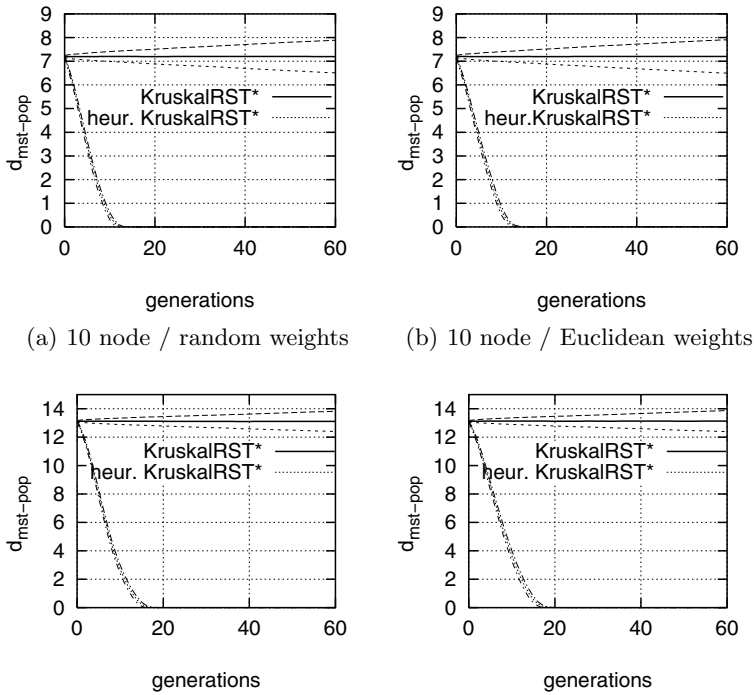
As before, we perform this experiment on 1000 randomly generated 10 and 16 node tree instances with random and Euclidean distance weights d_{ij}. For every tree instance we performed 50 runs with different, randomly chosen initial populations (KruskalRST) and 60 generations.

Figure 7.4 shows the mean and the standard deviation of $d_{mst-pop}$ over the number of generations. The plots compare the non-heuristic KruskalRST* crossover (p. 225) with the heuristic KruskalRST* crossover (p. 227). Only crossover and no selection is used. The results confirm the findings from Tzschoppe et al. (2004) and reveal that the crossover operator without heuristics is unbiased and does not modify the statistical properties of the population ($d_{mst-pop}$ remains constant over the number of generations). In contrast, the crossover operator with heuristics shows a strong bias towards the MST and the population quickly converges to the MST.

Mutation

Finally, we investigate the bias of the mutation operator for 1000 random tree instances of each type. As for the crossover operator we create a random population of 500 individuals using KruskalRST. Then, in every generation each individual is mutated exactly once using either the non-heuristic "insertion-before-deletion" mutation (p. 226) or the heuristic version (p. 227). Only mutation and no selection is used. For the heuristic mutation operator the parameter β is set to 1, 2, or 5. With lower β, edges with lower weights are preferred.

Figure 7.5 shows the mean and the standard deviation of $d_{mst-pop}$ over the number of generations. The results show that the non-heuristic mutation operator is unbiased, whereas the heuristic mutation is biased towards the MST. The bias increases with lower β. In contrast to the heuristic crossover operator, the population does not always converge completely towards the



(a) 10 node / random weights (b) 10 node / Euclidean weights

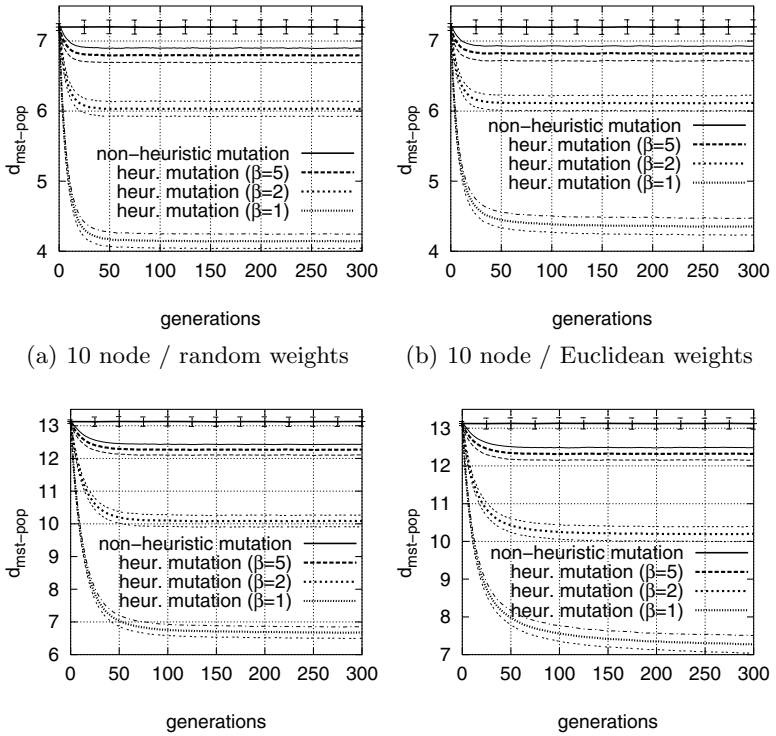
(c) 16 node / random weights (d) 16 node / Euclidean weights

Figure 7.4. The plots show the mean and the standard deviation of the distance $d_{mst-pop}$ between a population of 500 randomly generated individuals towards the MST over the number of generations. Only crossover and no selection is used. The results show that the non-heuristic KruskalRST* crossover is unbiased as the distance between the population and the MST remains constant. In contrast, the heuristic crossover operator is strongly biased towards the MST.

MST but the average distance of the population towards the MST remains stable after a few generations.

7.2.3 Performance for the OCST Problem

We want to investigate the performance of GEAs using the different variants of the edge-set encoding for the OCST problem from Sect. 8.2.1. The OCST problem is defined as follows: Let $G = (V, E)$ be a complete undirected graph with $n = |V|$ nodes and $|E|$ edges. To every pair of nodes (ij) a non-negative distance weight d_{ij} and a non-negative communication requirement r_{ij} is associated. The communication cost $c(T)$ (compare (8.2)) of a spanning tree T is defined as



(a) 10 node / random weights (b) 10 node / Euclidean weights

(c) 16 node / random weights (d) 16 node / Euclidean weights

Figure 7.5. The plots show the mean and the standard deviation of the distance $d_{mst-pop}$ between a population of 500 individuals towards the MST over the number of generations. Only mutation ("insertion before deletion") and no selection is used. The results show that the non-heuristic mutation operator is unbiased. The heuristic mutation operator is biased and the bias increases with lower β.

$$c(T) = \sum_{i,j \in V,\, i<j} r_{ij} \times d(p_{i,j}^T),$$

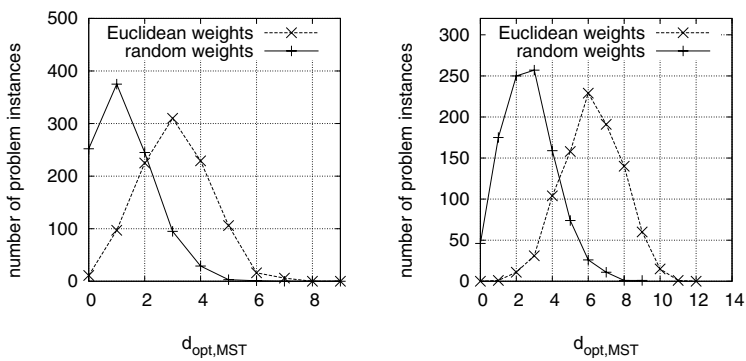
where $d(p_{i,j}^T)$ denotes the weight of the unique path from node i to node j in the spanning tree T. The OCST problem seeks the spanning tree with minimal costs among all other spanning trees. The OCST problem becomes the MST problem if there are no communication requirements r_{ij} and $c(T) = \sum_{e_{i,j} \in E} d_{ij}$.

It was shown in Rothlauf et al. (2003) that, on average, optimal solutions for OCST problems are similar to the MST. That means the average distance $d_{opt,MST}$ between the optimal solution T_{opt} and the MST is significantly lower than the average distance $d_{rnd,MST}^p$ between a randomly created tree and the MST.

Finding Optimal Solutions for OCST Problems

To investigate how the performance of the edge-set encoding depends on the structure of the optimal solution for a random OCST problem, an optimal or near-optimal solution must be determined. Due to the $\mathcal{NP}$-hardness of the OCST problem (compare Sect. 8.2.2), optimal solutions can be determined only for small problem instances with reasonable computational effort. Therefore, we limit our investigations to 10 and 16 node problem instances.

As GEA performance increases with N (Harik et al. 1997), to find a (near-optimal) solution we apply a GEA n_{iter} times to an OCST problem using a population size of N_0. T_0^{best} denotes the best solution of cost $c(T_0^{best})$ that is found during the n_{iter} runs. In the next round we double the population size and again apply a GEA n_{iter} times with a population size of $N_1 = 2N_0$. T_1^{best} denotes the best solution with cost $c(T_1^{best})$ that can be found in the second round. We continue this iteration and double the population size $N_i = 2N_{i-1}$ until $T_i^{best} = T_{i-1}^{best}$ and $n(T_i^{best})/n_{iter} > 0.5$, this means T_i^{best} is found in more than 50% of the runs in round i. $n(T_i^{best})$ denotes the number of runs that find the best solution T_i^{best} in round i.



(a) 10 node (b) 16 nodes

Figure 7.6. We randomly generated 1000 OCST problems and show the distribution of the problem instances over the distance $d_{opt,MST}$ between the optimal solution T_{opt} and the MST. Results are presented for 10 and 16 node problems using either random or Euclidean weights. The plots show that the optimal solutions for OCST problems are biased towards the MST.

For finding the optimal solutions we use a standard GA with traditional parameter settings. The problem was encoded using the NetKey representation (compare Sect. 6.5). The GA uses uniform crossover and tournament selection without replacement. The size of the tournament is three. The crossover probability is set to $p_{cross} = 0.8$ and the mutation probability (assigning a

random value $[0, 1]$ to one allele) is set to $p_{mut} = 0.02$. For the GA we started with $N_0 = 100$ and set $n_{iter} = 20$. Each GA run is stopped after a maximum of 200 generations. The computational effort for the experiments is high.

Figure 7.6 presents the results of our experiments. We show the number of problem instances over the distance $d_{opt,MST}$ between the optimal solution T_{opt} and the MST for 1000 randomly created OCST problems with 10 (Fig. 7.6(a)) and 16 (Fig. 7.6(b)) nodes. The OCST problems are created randomly using either random weights in]0,100] or placing the nodes randomly on a 1000×1000 two-dimensional grid and calculating the weights as the Euclidean distances between the nodes (details are described in Sect. 7.2.2). The demands r_{ij} between the nodes are random and uniformly distributed in]0,100].

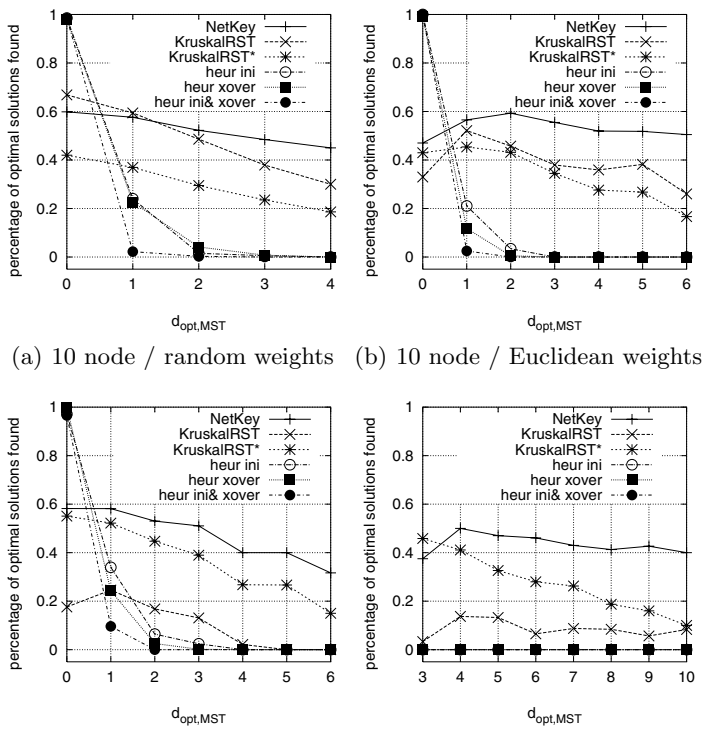
Comparing the results to the average distance $d^p_{rnd,MST}$ between a randomly created tree and the MST (Table 7.1) reveals that the optimal solutions for OCST problems are biased towards the MST. Furthermore, OCST problems with random weights show a stronger bias than OCST problems with Euclidean weights. Due to the bias of the optimal solutions towards the MST, the problem should be easy to solve for GEAs using the edge-set encoding.

The Performance of the Edge-Set Encoding for Randomly Generated OCST Problems

After determining optimal solutions as described in the previous paragraphs, we examine the performance of GEAs using the edge-set encoding. We use the same randomly generated problem instances as before and investigate how the GEA performance depends on the distance $d_{opt,MST}$ between the optimal solution and the MST. We use a generational GA with tournament selection without replacement of size two and no mutation. Each run is stopped after the population is fully converged or the number of generations exceeds 200. We perform 50 GA runs for each of the 1000 problem instances. In our experiments we compare the performance of GAs using

- non-heuristic KruskalRST crossover (p. 225) with non-heuristic KruskalRST initialization (p. 225) (indicated as "KruskalRST"),
- non-heuristic KruskalRST* crossover (p. 225) combined with non-heuristic KruskalRST initialization (indicated as "KruskalRST*"),
- non-heuristic KruskalRST* crossover combined with heuristic initialization (p. 226) with $\alpha = 1.5$ (indicated as "heur ini"),
- heuristic crossover (p. 227) combined with non-heuristic KruskalRST initialization (indicated as "heur xover"),
- heuristic crossover combined with heuristic initialization with $\alpha = 1.5$ (indicated as "heur ini & xover"), and
- as benchmark the unbiased network random key encoding with uniform crossover (indicated as "NetKey").

The population size N which is constant in all experiments, is chosen with respect to the performance of the non-heuristic KruskalRST* crossover operator. The aim is to find the optimal solution with a probability of about 50 %. Therefore, we choose for the 10 node problems a population size of $N = 60$ (random weights) and $N = 100$ (Euclidean weights) and for the 16 node problems a population size of $N = 200$ (random weights) and $N = 450$ (Euclidean weights).

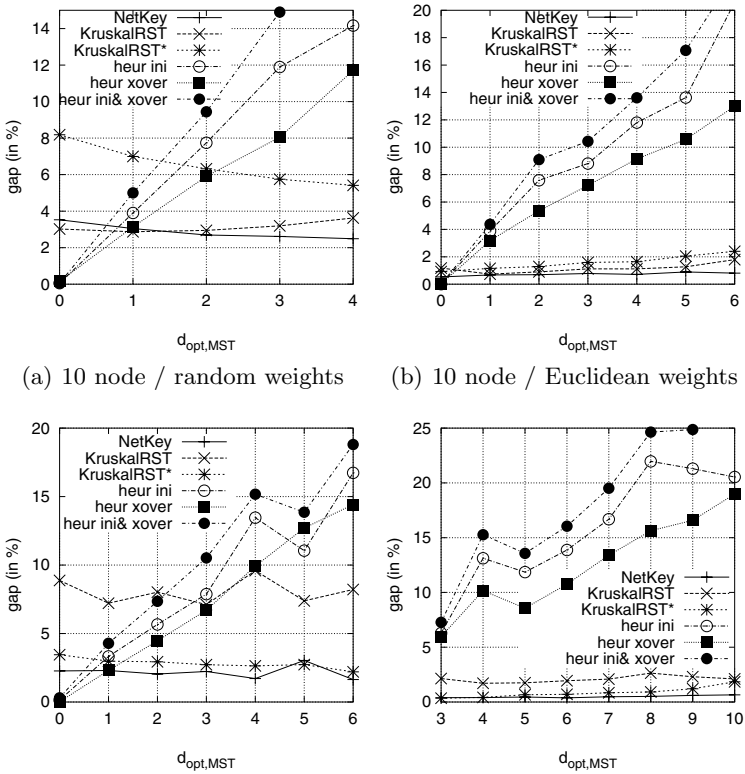


(a) 10 node / random weights (b) 10 node / Euclidean weights

(c) 16 node / random weights (d) 16 node / Euclidean weights

Figure 7.7. The figures compare the performance of a GA using different combinations of crossover and initialization operators for randomly generated 10 (left) and 16 (right) node OCST problems. The plots show the average percentage of optimal solutions that can be found over $d_{opt,MST}$. The heuristic crossover operator outperforms the non-heuristic version only if the optimal solution is very similar to the MST ($d_{opt,MST} \approx 0$). If $d_{opt,MST} > 1$ the heuristic crossover results in low GA performance. In contrast, when using the non-heuristic KruskalRST* crossover, GA performance remains about constant.

The results of the experiments are presented in Figs. 7.7 and 7.8. Figure 7.7 shows the percentage of GA runs that find the correct optimal solutions



(a) 10 node / random weights (b) 10 node / Euclidean weights

(c) 16 node / random weights (d) 16 node / Euclidean weights

Figure 7.8. We show the mean of the gap between the cost of the best found solution and the cost of the optimal solution over $d_{opt,MST}$. The results confirm that the heuristic crossover operator outperforms the non-heuristic variants only if the optimal solutions are very similar to the MST ($d_{opt,MST} \approx 0$).

over $d_{opt,MST}$. Fig. 7.8 shows the gap, $\frac{c(T_{found})-c(T_{opt})}{c(T_{opt})}$ (in percent), between the cost of the best found solution and the cost of the optimal solution over $d_{opt,MST}$. We show results for 1000 randomly generated problem instances. Results are plotted only for these $d_{opt,MST}$, where there are more than 10 problem instances. For example, we show results for 10 node problems with Euclidean weights only for $d_{opt,MST} \in \{0, \ldots, 6\}$ as there are only 8 (out of 1000) instances with $d_{opt,MST} = 7$ (compare Fig. 7.6(b)).

The results reveal that the heuristic crossover versions of the edge-set encoding (heur xover and heur ini & crossover) always find the optimal solution if the optimal solution is the MST. However for $d_{opt,MST} \neq 0$, the performance of GAs using the heuristic version drops sharply and the optimal solution can not be found if $d_{opt,MST} > 2$. In contrast, the performance of the non-heuristic KruskalRST* operator decreases only slightly with larger $d_{opt,MST}$ and allows

the GA to correctly identify the optimal solution even for larger $d_{opt,MST}$. The performance of the non-heuristic crossover combined with an heuristic initialization ("heur ini") is similar to the heuristic crossover operator. It always finds the optimal solution if it is the MST, however with increasing $d_{opt,MST}$ the decrease of performance is slightly less than for the heuristic crossover.

In summary, the heuristic crossover operator performs well only for problems where the optimal solution is slightly different from the MST. Otherwise, GAs using the edge-set encoding with heuristic crossover fail. The performance of GAs using the non-heuristic variant is similar to the performance of the NetKey encoding with uniform crossover. These results are confirmed when examining the gap $\frac{c(T_{found})-c(T_{opt})}{c(T_{opt})}$ (Fig. 7.8). Heuristic variants of the encoding show high performance if the optimal solution is the MST. However, with increasing $d_{opt,MST}$ the quality of the solutions strongly decreases and the non-heuristic variants outperform the heuristic variants.

In the remaining paragraphs, the performance of the edge-set-specific mutation operator is examined. As before 1000 random problems of different types are generated and the optimal solutions are calculated as described on page 232. For comparing the performance of different variants of the mutation operator, a simple simulated annealing (SA) strategy (van Laarhoven and Aarts 1988) is used as a representative example of mutation-based search. SA can be modeled as an GEA with population size one and Boltzmann selection (Mahfoud and Goldberg 1995). In each generation a new solution T_{off} is created by applying exactly one mutation to the parent solution T_{par}. If $c(T_{off}) < c(T_{par})$, T_{off} replaces T_{par}. If $c(T_{off}) > c(T_{par})$, T_{par} is replaced with probability $P(T) = \exp\left(-(c(T_{off}) - c(T_{par}))/T\right)$. With lower T, the probability of accepting worse solutions decreases.

In our experiments the start temperature $T_{start} = 50$ is reduced in every step by the factor 0.99. Therefore, $T_{t+1} = 0.99 \times T_t$. The number of search steps is set to $t_{max} = 300$ for 10 node and $t_{max} = 1000$ for 16 node problems. We performed 50 independent runs for each problem instance and investigated the performance of an SA using

- non-heuristic mutation (p. 226) and non-heuristic initialization (p. 225) (denoted as "no heur mut&ini"),
- non-heuristic mutation and heuristic initialization (p. 226) with $\alpha = 1.5$ (denoted as "no heur mut, $\alpha = 1.5$"),
- heuristic mutation (p. 227) with $\beta = 5$ and non-heuristic initialization (denoted as "$\beta = 5$, no heur ini"),
- heuristic mutation with $\beta = 5$ and heuristic initialization with $\alpha = 1.5$ (denoted as "$\beta = 5, \alpha = 1.5$"),
- heuristic mutation with $\beta = 0.5$ and non-heuristic initialization (denoted as "$\beta = 0.5$, no heur ini"), and
- heuristic mutation with $\beta = 0.5$ and heuristic initialization with $\alpha = 1.5$ (denoted as "$\beta = 0.5, \alpha = 1.5$").

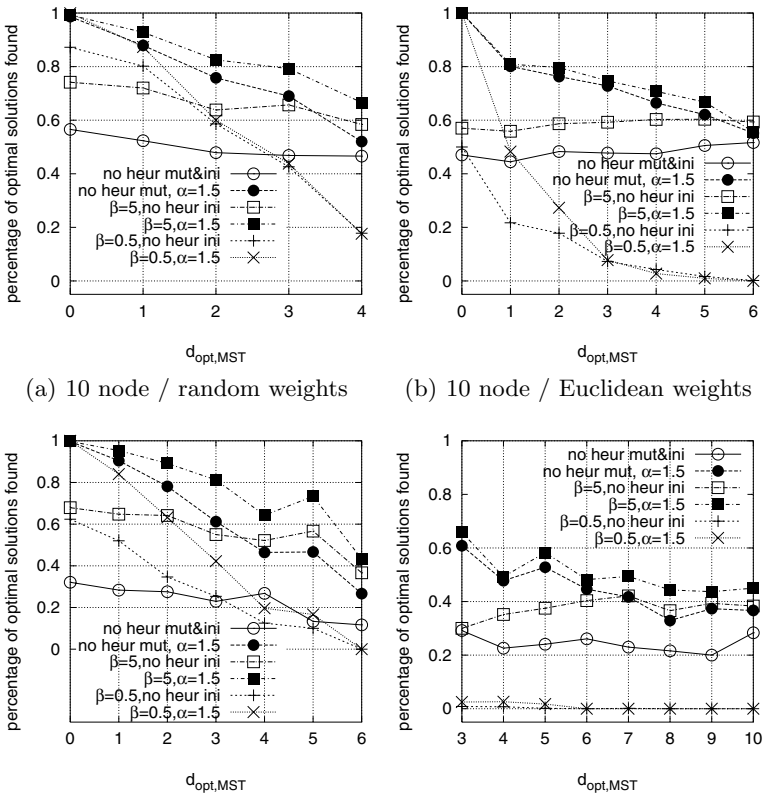
We performed no experiments for NetKeys as the corresponding mutation operator can not be directly compared. The mutation operator for NetKeys which changes one allele of the genotype, often does not change the corresponding phenotype, whereas the mutation operator of the edge-set encoding always changes one edge.

The results of the experiments are presented in Fig. 7.9. It shows the percentage of SA runs that find the correct optimal solutions over $d_{opt,MST}$. It can be seen that an SA using heuristic initialization always finds the optimal solution if $d_{opt,MST} = 0$. When using heuristic mutation with a low bias ($\beta = 5$), SA performance is always higher than when using non-heuristic mutation (for all considered $d_{opt,MST}$). A small bias of the mutation operator does not push the population towards the MST but allows a diversed population and efficient SA search for solutions somehow similar to the MST. However, when increasing the bias of the heuristic mutation to $\beta = 0.5$, SA performance becomes lower than for the non-heuristic case even for small $d_{opt,MST}$ (especially for the Euclidean problem instances). Then, the heuristic bias of the mutation operator is too strong and pushes the population too strongly towards the MST. The results reveal that by increasing the bias of the mutation operator (lowering β) problems where the optimal solutions are similar to the MST can be solved more efficiently; however, problems where the optimal solutions are different from the MST can be solved less efficiently.

To summarize our findings, the heuristic crossover operator of the edge-set encoding does not allow efficient search due to its strong bias towards the MST. Only problems where the optimal solutions are slightly different from the MST can be solved. The heuristic mutation operator results in good performance if β is large as the resulting low bias of the mutation operator prefers solutions similar to the MST and does not push a population too strongly towards the MST. However, if the bias towards the MST induced by β becomes stronger only optimal solutions similar to the MST can be found and mutation-based search fails. The results for the heuristic mutation operator show that the proper adjustment of β is important and crucial for the success of local search.

7.2.4 Summary and Conclusions

This section investigated the bias of the edge-set encoding which was proposed by Raidl and Julstrom (2003), and examines its performance for random instances of the optimal communication spanning tree (OCST) problem. The edge-set encoding belongs to the class of direct representations for trees. Instead of defining an additional genotype-phenotype mapping, encoding-specific initialization, crossover and mutation operators are directly applied to the trees. Section 7.2.1 described the functionality of the edge-set encoding and Sect. 7.2.2 performs an exhaustive investigation into the bias of the different variants of the edge-set encoding. The work is completed by an inves-



(a) 10 node / random weights (b) 10 node / Euclidean weights

(c) 16 node / random weights (d) 16 node / Euclidean weights

Figure 7.9. The figures show the performance of an SA using different variants of initialization and mutation operators of the edge-set encoding. The plots show the average percentage of optimal solutions that can be found over $d_{opt,MST}$ for 1000 randomly created OCST problems. The results show that the heuristic variants of the mutation operator outperform the non-heuristic variants for the OCST problem if β is set properly.

tigation into the performance of crossover-based search and mutation-based search for randomly generated instances of the OCST problem.

The investigation into the bias of the edge-set encoding reveals that the heuristic versions of the initialization, crossover, and mutation operators are biased towards the MST defined on the distance weights. The bias is especially strong for the heuristic crossover operator which results in a quick convergence of a population of trees towards the MST. In contrast, the non-heuristic search operators of the edge-sets are unbiased and their application results in an undirected and uniform search through the search space.

Due to the strong bias of the heuristic search operators towards the MST, tree optimization problems can easily be solved if optimal solutions are the MST. However, if optimal solutions are only slightly different from the MST, the heuristic crossover operator fails due to its strong bias towards the MST. Therefore, the heuristic crossover operator is not appropriate for solving tree optimization problems. Rothlauf and Tzschoppe (2004) proposed an extension of the heuristic crossover operator which allows us to reduce its strong bias. Using the modified heuristic crossover operators results in higher GEA performance if the optimal solution is different from the MST. In contrast to the heuristic crossover operator, the non-heuristic crossover operator of the edge-sets results in good performance for OCST problems even if the optimal solutions are quite different from the MST. Its performance is similar to the NetKey encoding from Sect. 6.5. For the mutation operator, the strength of the bias towards the MST can be controlled by an encoding-specific parameter β. With high β, the bias towards the MST is low, with low β it is strong. Therefore, with low β, tree problems can be solved more efficiently if the optimal solutions are similar to the MST, but otherwise less efficiently. If β is set appropriately, the heuristic mutation operator is a good choice for OCST problems as optimal solutions of this problem are similar to the MST.

The problems of the heuristic variants of the edge-set encoding emphasize the difficulties of a proper design of direct representations. In contrast to indirect encodings, the behavior of new, problem-specific search operators is often unknown. The analysis of the edge-set encoding has shown that although optimal solutions for the OCST problems are biased towards the MST (Rothlauf et al. 2003), direct representations such as the heuristic edge-set encoding that use this problem-specific knowledge and are biased towards the MST, can fail if the bias is too strong. Therefore, unbiased representations should be used if no problem-specific knowledge is known a priori.

8

Performance of Genetic and Evolutionary Algorithms on Tree Problems

In the previous chapters, the presented theory about representations was used for the analysis and design of representations as well as search operators. The investigations into the properties of representations were based on theory and helped us to understand what happens when GEAs use a specific representation. However, in practice, GEA users are often less interested in theory about representations but want simple instruments for a quick and rough prediction of the expected performance of a representation. They have several representations at hand and want to know which representation they should choose for their problem. We do not want to leave them alone with their problems, but illustrate how they can advantageously use the proposed theory.

This chapter illustrates for scalable test tree problems and real-world tree problems how the performance of GEAs using different types of representations can be predicted by using the provided framework about representations. Based on the framework, we give qualitative predictions of solution quality and time to convergence for different types of tree representations. Doing this, this chapter also provides an exhaustive comparison of the performance of different tree representations. For our comparison we choose the indirect representations Prüfer numbers (Sect. 6.2), characteristic vectors (Sect. 6.3), the link and node biased encoding (Sect. 6.4), and NetKeys (Sect. 6.5) as well as the direct encodings NetDir (Sect. 7.1) and edge-sets (Sect. 7.2). These indirect and direct representations are used on scalable test problems like the one-max tree and deceptive trap problem for trees and on various test instances of the optimal communication spanning tree (OCST) problem from the literature. The results show that using the outlined theory makes it easier to select the proper representation for the problem at hand.

The test instances of the OCST problem are chosen because the exact specifications of the problems are either easily available (Raidl 2001; Rothlauf et al. 2002) or published (Palmer 1994; Berry et al. 1995). For summarizing purposes, the exact data regarding the distance weights and the communication demands for the test instances of the OCST problems are listed in Appendix A.

The following section provides a comparison of GEA performance for scalable test problems. After a brief analysis of representations in Sect. 8.1.1, we present in Sect. 8.1.2 results for the fully easy one-max tree problem and in Sect. 8.1.3 results for the the fully difficult deceptive tree problem. For each of the two test problems, we provide theoretical predictions and empirical evidence. Then, we focus in Sect. 8.2 on the OCST problem. This problem is defined on trees and researchers have proposed some test instances in the literature (Palmer 1994; Berry et al. 1995; Raidl 2001; Rothlauf et al. 2002). For each problem, we deduce predictions of GEA performance and present empirical results. The chapter ends with a brief summary.

8.1 GEA Performance on Scalable Test Tree Problems

This section compares the performance of different types of indirect and direct representations for the one-max tree problem and the deceptive trap problem for trees. After a brief analysis of tree representations based on the framework from Chap. 4, we present results for the one-max tree problem (Sect. 8.1.2) and for the deceptive tree problem (Sect. 8.1.3). For both problems, we provide brief descriptions, theoretical predictions about GEA performance, and empirical evidence.

8.1.1 Analysis of Representations

We briefly summarize the most important properties of the different tree representations and operators from the previous chapters. We focus on the results concerning redundancy, bias, scaling, and locality for Prüfer numbers, characteristic vectors (CV), the LNB encoding, NetKeys, NetDir, and edge-sets.

For the use of Prüfer numbers, NetKeys, CV, and NetDir no additional representation-specific parameters are necessary. In contrast, for LNBs and edge-sets, additional representation-specific parameters must be set by the user. In the following experiments, we use the same variants of the edge-sets as in the investigation presented in Sect. 7.2.3:

- **KruskalRST**: non-heuristic KruskalRST crossover (p. 225) with non-heuristic KruskalRST initialization (p. 225),
- **KruskalRST***: non-heuristic KruskalRST* crossover (p. 225) combined with non-heuristic KruskalRST initialization,
- **heur ini**: non-heuristic KruskalRST* crossover combined with heuristic initialization (p. 226) with $\alpha = 1.5$,
- **heur xover**: heuristic crossover (p. 227) combined with non-heuristic KruskalRST initialization, and
- **h ini & xover**: heuristic crossover combined with heuristic initialization with $\alpha = 1.5$.

Further details about the edge-set encoding can be found in Sect. 7.2. For the LNB encoding, we use the following variants:

- **NB** (P_2=1): the node-biased encoding (p. 180) with the node-specific bias $P_2 = 1$,
- **NB** (P_2=20): the node-biased encoding with the node-specific bias $P_2 = 20$,
- **LB** (P_1=1): the link-biased encoding (p. 181) with the link-specific bias $P_1 = 1$,
- **LB** (P_1=20): the link-biased encoding with the link-specific bias $P_1 = 20$, and
- **LNB** (P_1=P_2=1): the link-and-node-biased encoding (p. 182) with the link-specific bias $P_1 = 1$ and the node-specific bias $P_2 = 1$,

The following paragraphs review important properties of the representations presented in Chaps. 6 and 7 which are summarized in Table 8.1.

The investigation into the redundancy of indirect representations revealed that Prüfer numbers are non-redundant and that NetKeys are uniformly and synonymously redundant. Furthermore, the LNB encodings are synonymously redundant; however, the over- and under-representation of solutions depends on the setting of the node-specific and link-specific bias. Variants of the LNB encoding that use a large link-specific bias P_1 are approximately uniformly redundant and show the same behavior as NetKeys. Therefore, GEA performance is independent of the structure of the optimal solution. In contrast, the LNB encoding is non-uniformly redundant if the link-specific bias is not large enough. If P_1 and P_2 are small, LNB encodings are biased towards the minimum spanning tree. For a large node-specific bias P_2, the encoding is biased towards stars. Therefore, GEA performance depends on the structure of the optimal solution. In contrast, CVs are uniformly redundant but affected by non-synonymous redundancy. As not all genotypes that represent the same tree are similar to each other, the recombination of two genotypes that encode similar phenotypes can result in an offspring with different properties (compare Sect. 6.3.3).

The situation is different for indirect encodings such as NetDir or edge-sets. For direct encodings, it is not the properties of the genotype-phenotype mapping which are relevant but the properties of the search operators. The investigation into the NetDir encoding shows (compare Sect. 7.1) that the search operators are nearly unbiased which results in the same GEA performance as with non-redundant or non-uniformly redundant encodings. In contrast, the edge-sets show a bias towards the minimum spanning tree if heuristic variants of the initialization and search operators are used. A biased search operator is equivalent to a non-uniformly redundant encoding. Due to the design of the search operator, both direct representations show high locality as the application of a mutation operator always results in a similar solution.

The investigation into the locality of tree representations shows that the Prüfer number representation has in general low locality. However, the locality

Table 8.1. Summary of important properties of direct and indirect representations for trees

	redundancy & bias	locality	BB-scaling
Prüfer	non-redundant	high locality around stars, low locality elsewhere	uniformly scaled
NetKey	uniformly and synonymously redundant	high locality	uniformly scaled
CV	uniformly and non-synonymously redundant	low locality due to non-synonymous redundancy	uniformly scaled
NB (P_2=1 and P_2=20)	non-uniformly (bias towards stars) and synonymously redundant	high locality	uniformly scaled
LB (P_1=1)	non-uniformly (bias towards MST) and synonymously redundant	high locality	uniformly scaled
LB (P_1=20)	non-uniformly and synonymously redundant	high locality	uniformly scaled
LNB (P_1=P_2=1)	non-uniformly (bias towards stars and MST) and synonymously redundant	high locality	uniformly scaled
NetDir	non-redundant, search operator unbiased	high locality	-
KruskalRST	non-redundant, search operator unbiased	high locality	-
KruskalRST*	non-redundant, search operator unbiased	high locality	-
heur. ini	non-redundant, initialization biased towards MST (equivalent to non-uniform redundancy)	high locality	-
heur. xover	non-redundant, search operator biased towards MST (equivalent to non-uniform redundancy)	high locality	-
h. ini & xover	non-redundant, initialization and search operator biased towards MST (equivalent to non-uniform redundancy)	high locality	-

of Prüfer numbers is not low everywhere. When encoding stars, the locality of Prüfer numbers is high; when encoding non-stars, the locality of Prüfer numbers is low. For redundant representations, low locality is equivalent to non-synonymity. As for low-locality representations, similar genotypes do not encode similar phenotypes if non-synonymously redundant representations are used. Therefore, all encodings that are synonymously redundant can be viewed as high-locality encodings since small changes in the genotype always result in small changes in the phenotype. The non-synonymously redundant CV encoding has low locality as small changes of the genotype do not necessarily

result in a similar phenotype. For direct encodings, the locality is always high as the mutation operators are designed such that trees with similar properties are created.

The investigation into the scaling of the BBs has revealed that all examined indirect tree representations have uniformly scaled BBs. There are no BBs that have a higher contribution to the fitness of a tree. Therefore, the dynamics of genetic search are not changed and all alleles are solved implicitly in parallel.

Table 8.2. We generated random genotypes x^g_{rnd} of different sizes n and calculated the minimum phenotypic distance $\min(d^p_{rnd,star})$ to stars, and the phenotypic distance $d^p_{rnd,MST}$ to the MST. The numbers confirm that CVs, NetKey, NetDir, LB ($P_1 = 20$), and edge-sets without initialization heuristics are unbiased (uniformly redundant) as the distances are about the same as for the non-redundant Prüfer numbers. Edge-sets with heuristic initialization are biased towards the MST. The LNB encoding is biased towards stars for large P_2 and towards the MST for low P_1 and P_2.

problem size n		8	20	40
distance d		μ (σ)	μ (σ)	μ (σ)
Prüfer number	$\min(d^p_{rnd,star})$	3.66 (0.64)	14.67 (0.79)	34.07 (0.84)
	$d^p_{rnd,MST}$	5.17 (1.01)	17.05 (1.26)	37.03 (1.34)
NetKey	$\min(d^p_{rnd,star})$	3.75 (0.62)	14.77 (0.78)	34.15 (0.80)
	$d^p_{rnd,MST}$	5.22 (1.01)	17.10 (1.25)	37.06 (1.32)
CV	$\min(d^p_{rnd,star})$	3.66 (0.64)	14.68 (0.80)	34.07 (0.83)
	$d^p_{rnd,MST}$	5.18 (1.01)	17.05 (1.27)	37.02 (1.34)
NB (P_2=1)	$\min(d^p_{rnd,star})$	2.46 (1.1)	10.72 (2.26)	26.90 (3.14)
	$d^p_{rnd,MST}$	3.31 (1.05)	11.65 (1.77)	26.85 (2.50)
NB (P_2=20)	$\min(d^p_{rnd,star})$	0.24 (0.68)	1.64 (2.79)	5.98 (7.09)
	$d^p_{rnd,MST}$	5.08 (0.80)	16.55 (1.13)	35.88 (1.53)
LB (P_1=1)	$\min(d^p_{rnd,star})$	3.74 (0.63)	14.79 (0.76)	34.24 (0.76)
	$d^p_{rnd,MST}$	3.54 (1.15)	12.64 (1.86)	28.93 (2.53)
LB (P_1=20)	$\min(d^p_{rnd,star})$	3.74 (0.61)	14.77 (0.76)	34.14 (0.80)
	$d^p_{rnd,MST}$	5.15 (1.06)	16.79 (1.34)	36.36 (1.53)
LNB (P_1=P_2=1)	$\min(d^p_{rnd,star})$	3.06 (0.89)	12.55 (1.47)	30.18 (1.96)
	$d^p_{rnd,MST}$	3.90 (1.07)	13.88 (1.69)	31.74 (2.21)
NetDir	$\min(d^p_{rnd,star})$	3.66 (0.65)	14.67 (0.79)	34.07 (0.83)
	$d^p_{rnd,MST}$	5.17 (1.02)	17.05 (1.26)	37.02 (1.34)
KruskalRST, KruskalRST*, heur xover	$\min(d^p_{rnd,star})$	3.75 (0.61)	14.78 (0.77)	34.17 (0.81)
	$d^p_{rnd,MST}$	5.26 (1.04)	17.10 (1.27)	37.04 (1.33)
heur ini, heur ini & xover	$\min(d^p_{rnd,star})$	4.06 (0.52)	15.75 (0.45)	35.53 (0.53)
	$d^p_{rnd,MST}$	0.35 (0.58)	1.13 (1.29)	2.29 (2.30)

To investigate and verify the bias of the different representations, we can measure the average distance of randomly generated solutions to a star or to the MST. Consequently, Table 8.2 presents results for randomly created genotypes that encode trees of different size n. For each representation, we

randomly generate 10,000 genotypes and measure the mean μ and the standard deviation σ of the minimum phenotypic distance $\min(d^p_{rnd,star})$ towards a star, and the phenotypic distance $d^p_{rnd,MST}$ towards the MST. For the different variants of the LNB encoding, we randomly placed the n nodes on a two-dimensional grid of size 1,000×1,000. As the distance weights d_{ij} are relevant for the construction of the phenotype (compare (6.5)), we used Euclidean distances between the nodes i and j as distance weights d_{ij}. For the edge-sets, we created 100 random populations of size $N = 100$, and placed for each population the nodes randomly on the 1,000×1,000 square.

The results confirm the findings from above. The direct encodings NetDir as well as variants of the edge-sets that use non-heuristic initialization (KruskalRST, KruskalRST*, and heur. xover) are about unbiased as the distances are similar to Prüfer numbers. Also, the uniformly redundant encodings NetKeys, CVs, and the LB encoding with large link-specific bias are unbiased because they have about the same distances as non-redundant representations (Prüfer numbers). In contrast, the NB and LNB encoding show a bias towards stars as $\min(d^p_{rnd,star})$ is lower in comparison to the unbiased Prüfer numbers. Variants of the edge-sets that use heuristic initialization as well as the LB ($P_1 = 1$) and LNB encoding show a bias towards the MST as $d^p_{rnd,MST}$ is lower in comparison to Prüfer numbers.

8.1.2 One-Max Tree Problem

We examine the performance of GAs using different representations for the one-max tree problem introduced in Sect. 6.1.5.

Problem Description

The one-max tree problem was defined in Sect. 6.1.5 as a fully easy tree problem. The problem is fully easy for GEAs as the phenotypic size of the BBs is $k_p = 1$. All tree schemata that contain the optimal solution have higher fitness than their competitors. For the one-max tree problem, an optimal solution T_{opt} is chosen a priori either randomly or by hand. The structure of T_{opt} can either be a random tree, a star, a list, the MST, or any other pre-defined tree structure. In the following section, we assume a minimization problem and define the fitness of a tree T_i as the distance $d_{i,opt}$ between the optimal solution and the tree. For $d_{i,opt} = 0$, $T_i = T_{opt}$.

Further details regarding the one-max tree problem are provided in Sects. 6.1.5 and 6.1.4.

Theoretical Predictions

We give predictions on GEA performance for the one-max tree problem. The predictions are based on the results from Sect. 8.1.1.

We know that some representations are non-uniformly redundant and are biased towards specific tree structures. Therefore, GEA performance depends on the structure of the optimal solution. Using the results learnt about redundancy from Sect. 8.1.1, we expect the CV, NetKey, NetDir, edge-set without heuristics, and LB ($P_1 = 20$) encoding to perform independently of the structure of the optimal tree T_{opt}. GEAs using LNB encodings with small biases P_1 and P_2 will perform better if the optimal solutions are similar to the MST. However, GEAs using these encodings need more generations and find less BBs if the optimal solutions are not similar to the MST. GEAs using the NB encoding are biased towards stars and show high performance if the best solutions are stars. If the best solutions are arbitrary trees or lists, GEAs will show lower performance when using the LNB encoding. The edge-set encoding with heuristics introduce a bias towards MST-like solutions. Therefore, GEAs using edge-sets with heuristics are expected to perform well for problems where the optimal solutions are the MST, and show low performance elsewhere.

After we have examined the effects of redundancy, we focus on locality. We have seen that Prüfer numbers have high locality around stars, but low locality elsewhere. This means that the genotypic size of the BBs k_g is larger than the phenotypic size of the BBs $k_p = 1$ if the optimal solution is not a star. Therefore, the performance of GEAs using Prüfer numbers is low if the optimal solution is not a star. CVs are also affected by low locality as the encoding is non-synonymously redundant. Due to stealth mutation, offspring can be created that are not similar to their parents. In general, we expect that GEAs using CVs show lower performance for the easy one-max problem, as the search is randomized and guided search is no longer possible. However, as we use only crossover and no mutation in our experiments, BBs can not come back into the population once they are extinct. The non-synonymous redundancy of the CV encodings can partially solve this problem as it works like stealth mutation (compare Sect. 6.3.3) and brings back lost BBs. Therefore, for small problem instances, we expect GEAs using the CV encoding to be able to find more BBs than other encodings, but to need much more time to do this. The run duration and the number of solved problems at the end of the run increases when using CVs. This effect of stealth mutation is amplified by the low difficulty of the one-max problem. This problem is especially easy for mutation-based GEAs because the landscape leads GEAs to the correct solution. Therefore, mutation, as well as stealth mutation, increases GEA performance for small problem instances.

In contrast, the other representations (NetKey, NetDir, edge-sets and LNB) have high locality and the problem difficulty is not changed.

Empirical Results

For our experiments, we use a simple, generational genetic algorithm without mutation, tournament selection without replacement of size 2, and uniform crossover. We randomly generate 200 problems for each problem instance and

perform for each problem instance and each representation 25 GA runs. We present results for the one-max tree problem with 8, 20, and 40 nodes. The optimal solution is either a randomly chosen star or list, an arbitrary tree, or the MST based on the the distance weights d_{ij}. The performance of GEAs is determined by the percentage P_{succ} of correctly solved problems at the end of the run and the number of generations t_{conv} until the population is fully converged. We use a population size of $N = 16$ for all $n = 8$ node problems, $N = 100$ for all $n = 20$ node problems, and $N = 600$ for all $n = 40$ node problems. The population sizes N are chosen such that some of the representations allow the GA to always find the optimal solutions. In general, a higher population size N increases the performance (higher P_{succ}), whereas lower N reduces GA performance. The performance differences between the different representations remain about the same when using different population sizes. All runs are stopped after the population is fully converged or a maximum of 200 generations is reached.

The performance of GAs using different representations (compare Sect. 8.1.1) for the one-max tree problem is shown in Tables 8.3 and 8.4. We present results for different optimal solutions (T_{opt} is either an arbitrary tree, the MST, a random star, or a random list) and show the percentage P_{succ} of runs that find the optimal solution T_{opt}, the mean and standard deviation of the fitness of the best found solution, and the mean and standard deviation of the number of generations t_{conv} until the population is fully converged or the GA run is stopped after 200 generations. The fitness of the best found solution directly measures the distance between the optimal solution T_{opt} and the best found solution. All results are averaged over 200 randomly created problems and 25 runs for each problem.

The numbers show that the performance of GEAs searching for optimal lists is about the same as when searching for optimal arbitrary trees. Furthermore, the performance of GAs using uniformly redundant encodings like CVs, NetKeys, NetDir, edge-sets without heuristics, or the LB ($P_1 = 20$) encoding is about independent of the structure of the optimal solution T_{opt}. There are some exceptions for T_{opt} is a star, but, in general, P_{succ}, the fitness, and T_{conv} are not affected by the optimal solution being an arbitrary tree, the MST, a star, or a list.

The performance of GAs using Prüfer number is low due to the low locality of the encoding. However, we have seen in Sect. 6.2.4 that the locality of Prüfer number is high around stars. Therefore, GAs using Prüfer numbers perform better when searching for optimal stars than when searching for optimal trees, lists, or MSTs.

If the problem instances are small ($n = 8$ and $n = 20$), GAs using the CV encoding are able to find the optimal solution when searching for an arbitrary tree, a list, or the MST, but need a high number of generations t_{conv}. The non-synonymous redundancy of the encoding continuously introduces new genetic material into the search and prolongs search time allowing the GA to find its way to the optimal solutions. However, for larger problem instances, the search

Table 8.3. Performance of GAs using different types of representations for one-max tree problems of different sizes and with different T_{opt} (arbitrary tree and MST)

T_{opt}		8 nodes			20 nodes			40 nodes		
		P_{succ}	fitness μ (σ)	t_{conv} μ (σ)	P_{succ}	fitness μ (σ)	t_{conv} μ (σ)	P_{succ}	fitness μ (σ)	t_{conv} μ (σ)
arbitrary tree	Prüfer	0.05	1.65 (0.8)	18 (4)	0	5.91 (1.2)	77 (9)	0	14.28 (1.7)	198 (5)
	NetKey	0.59	0.47 (0.6)	16 (3)	0.91	0.09 (0.3)	40 (4)	1	0 (0)	85 (9)
	CV	0.95	0.05 (0.2)	20 (4)	1	0.01 (0.1)	137 (22)	0	17.34 (1.0)	200 (0)
	NB (P_2=1)	0	3.20 (0.9)	10 (3)	0	12.44 (1.4)	28 (6)	0	29.58 (1.8)	62 (12)
	NB (P_2=20)	0	3.52 (0.7)	7 (3)	0	13.18 (1.1)	20 (5)	0	30.06 (1.5)	38 (9)
	LB (P_1=1)	0.11	1.63 (0.9)	14 (4)	0	5.72 (1.8)	51 (12)	0	14.73 (2.7)	150 (32)
	LB (P_1=20)	0.58	0.47 (0.6)	16 (3)	0.90	0.11 (0.3)	40 (4)	1	0 (0)	85 (9)
	P_1=P_2=1	0.13	1.47 (0.9)	18 (4)	0	4.51 (1.5)	69 (11)	0	12.22 (2.4)	190 (15)
	NetDir	0.39	0.79 (0.7)	20 (6)	0.72	0.32 (0.5)	125 (35)	0	10.08 (0.8)	200 (0)
	KrukalRST	0.40	0.76 (0.7)	25 (9)	0.02	2.14 (0.9)	200 (0)	0	13.75 (0.9)	200 (0)
	KruskalRST*	0.21	1.15 (0.8)	11 (2)	0.54	0.58 (0.7)	26 (2)	0.99	0.01 (0)	38 (1)
	heur. ini	0	4.89 (1.1)	2 (2)	0	16.55 (1.4)	3 (3)	0	36.43 (1.5)	4 (3)
	heur. xover	0	3.28 (0.9)	9 (3)	0	13.60 (1.5)	17 (4)	0	33.01 (1.8)	20 (4)
	h. ini & xover	0	5.19 (1.0)	1 (2)	0	16.98 (1.3)	3 (3)	0	36.88 (1.4)	3 (3)
MST	Prüfer	0.03	1.75 (0.7)	18 (4)	0	6.15 (1.1)	79 (8)	0	14.44 (1.6)	199 (2)
	NetKey	0.57	0.48 (0.6)	16 (3)	0.91	0.10 (0.3)	39 (4)	1	0 (0)	81 (8)
	CV	0.95	0.05 (0.2)	20 (4)	1	0 (0)	132 (19)	0	16.76 (1.0)	200 (0)
	NB (P_2=1)	0.55	0.50 (0.5)	11 (3)	0.39	0.82 (0.7)	37 (5)	0.51	0.65 (0.5)	95 (12)
	NB (P_2=20)	0	2.96 (0.8)	10 (3)	0	7.19 (1.3)	26 (5)	0	7.79 (1.6)	64 (10)
	LB (P_1=1)	0.96	0.04 (0.2)	12 (2)	1	0 (0)	34 (4)	1	0 (0)	84 (9)
	LB (P_1=20)	0.61	0.43 (0.6)	16 (3)	0.93	0.07 (0.2)	39 (4)	1	0 (0)	81 (8)
	P_1=P_2=1	0.78	0.23 (0.4)	17 (4)	0.87	0.14 (0.3)	63 (8)	0.98	0.02 (0.0)	181 (16)
	NetDir	0.37	0.82 (0.7)	20 (6)	0.73	0.33 (0.6)	167 (26)	0	9.96 (0.8)	200 (0)
	KrukalRST	0.42	0.73 (0.7)	25 (9)	0.02	1.98 (0.8)	200 (0)	0	13.54 (0.9)	200 (0)
	KruskalRST*	0.23	1.12 (0.8)	11 (2)	0.59	0.51 (0.7)	25 (2)	1	0 (0)	36 (1)
	heur. ini	1	0 (0)	2 (1)	1	0 (0)	4 (1)	1	0 (0)	5 (0)
	heur. xover	0.65	0.39 (0.6)	6 (1)	0.97	0.03 (0.1)	11 (1)	1	0 (0)	14 (0)
	h. ini & xover	1	0 (0)	1 (0)	1	0 (0)	2 (0)	1	0 (0)	3 (0)

space becomes too large and the random search behavior of GAs using the CV encoding does not allow it to find the optimal solution any more. When searching for optimal stars, the performance of GAs using the CV encoding is in general low as the repair process has problems with creating star-like structures.

LB encodings with $P_2 = 20$ are strongly biased towards stars and GAs perform very well when searching for the optimal solution is a star, but fail completely when searching for lists, arbitrary trees, or the MST. When using the LB encoding with $P_2 = 1$, the bias towards the star is smaller and there is an additional bias towards the MST. Therefore, this variant of the LB encoding performs slightly better for non-stars especially if the optimal solution is the MST. If the LB encodings only use a small link-specific bias $P_1 = 1$, the

Table 8.4. Performance of GAs using different types of representations for one-max tree problems of different sizes and with different T_{opt} (star and list)

T_{opt}		8 nodes			20 nodes			40 nodes		
		P_{succ}	fitness μ (σ)	t_{conv} μ (σ)	P_{succ}	fitness μ (σ)	t_{conv} μ (σ)	P_{succ}	fitness μ (σ)	t_{conv} μ (σ)
random star	Prüfer	0.46	0.72(0.8)	14 (3)	0.72	0.54 (1.0)	43 (9)	0.98	0.02 (0.1)	73 (18)
	NetKey	0.67	0.38(0.6)	16 (3)	0.94	0.06 (0.2)	41 (5)	1	0 (0)	94 (9)
	CV	0.94	0.08(0.4)	20 (4)	0	8.03 (0.9)	200 (0)	0	27.49(0.8)	200 (0)
	NB (P_2=1)	0.57	0.95(1.4)	10 (3)	0.85	0.28 (0.8)	26 (7)	0.98	0.03 (0.1)	47 (10)
	NB (P_2=20)	0.90	0.66(1.9)	6 (2)	1	0.02 (0.3)	13 (3)	1	0 (0)	27 (5)
	LB (P_1=1)	0.23	1.68(1.4)	14 (4)	0.07	5.78 (3.5)	52 (13)	0.02	15.32(7.7)	140(38)
	LB (P_1=20)	0.66	0.38(0.6)	16 (3)	0.94	0.06 (0.3)	41 (5)	1	0 (0)	95 (9)
	P_1=P_2=1	0.77	0.34(0.8)	16 (4)	0.99	0.01 (0.1)	48 (7)	1	0 (0)	135(16)
	NetDir	0.30	1.05(0.9)	17 (6)	0.48	0.72 (0.8)	56 (34)	0.99	0.01 (0.1)	64 (4)
	KrukalRST	0.26	1.07(0.8)	24(11)	0.24	1.22 (1.0)	127(41)	0	20.84 (1)	200 (0)
	KruskalRST*	0.14	1.42(0.9)	11 (2)	0.14	1.65 (1.1)	30 (2)	0.69	0.36 (0.6)	54 (2)
	heur. ini	0	4.94(0.9)	2 (2)	0	16.58(1.0)	3 (4)	0	36.47(1.1)	4 (4)
	heur. xover	0	3.64(1.1)	9 (3)	0	14.41(1.4)	17 (4)	0	33.99(1.5)	20 (4)
	h. ini & xover	0	5.23(0.7)	1 (2)	0	17.05(0.7)	2 (3)	0	36.96(0.7)	3 (4)
random list	Prüfer	0.02	1.81(0.7)	18 (4)	0	6.20 (1.1)	80 (9)	0	14.35(1.7)	199 (2)
	NetKey	0.55	0.51(0.6)	16 (3)	0.90	0.11 (0.3)	39 (4)	1	0 (0)	76 (7)
	CV	0.96	0.04(0.2)	20 (4)	1	0 (0)	114(14)	0	16.31(0.9)	200 (0)
	NB (P_2=1)	0	3.61(0.8)	11 (3)	0	13.04(1.3)	29 (6)	0	30.20(1.6)	65 (12)
	NB (P_2=20)	0	4.51(0.6)	7 (4)	0	14.23(1.0)	25 (5)	0	30.67(1.4)	47 (9)
	LB (P_1=1)	0.07	1.65(0.8)	14 (4)	0	5.68 (1.6)	51 (12)	0	14.47(2.5)	151(31)
	LB (P_1=20)	0.54	0.51(0.6)	16 (3)	0.88	0.13 (0.3)	39 (4)	1	0 (0)	77 (8)
	P_1=P_2=1	0.03	1.83(0.8)	19 (4)	0	6.12 (1.3)	70 (11)	0	15.82(2.0)	191(14)
	NetDir	0.37	0.80(0.7)	21 (6)	0.63	0.48 (0.7)	198(10)	0	9.94 (0.8)	200 (0)
	KrukalRST	0.43	0.71(0.7)	25 (9)	0.01	1.93 (0.8)	200 (0)	0	13.76(0.9)	200 (0)
	KruskalRST*	0.25	1.07(0.8)	11 (2)	0.63	0.44 (0.6)	25 (2)	1	0 (0)	35 (1)
	heur. ini	0	4.90(1.1)	2 (2)	0	16.53(1.4)	3 (3)	0	36.43(1.5)	4 (3)
	heur. xover	0	3.23(1.0)	9 (3)	0	13.49(1.6)	17 (4)	0	33.11(1.9)	20 (4)
	h. ini & xover	0	5.19(1.1)	2 (2)	0	16.97(1.3)	3 (3)	0	36.84(1.4)	3 (3)

encoding is biased towards MSTs. Therefore, GEAs using this encoding show low performance when searching for stars, lists, or arbitrary trees but high performance when searching for the MST. When using the LB encoding with a large link-specific bias $P_1 = 20$ the encoding performs the same as NetKeys. When using the LNB encoding with $P_1 = P_2 = 1$, there is a bias towards stars and the MST, and GAs show a high performance when searching for optimal stars or for the MST. However, for optimal lists or arbitrary trees, GA performance is low.

The performance of the direct encodings NetDir and edge-sets is more difficult to predict in comparison to the indirect encodings as the design of the search operator is problem-specific, and it is often not obvious whether there is a bias. The results for the NetDir encoding show that the crossover

operator is biased towards stars. If the optimal solution is an arbitrary tree, a list, or the MST, GAs using the NetDir encoding fail for larger problem instances ($n = 40$). The results for the edge-sets show that GAs using the non-heuristic KruskalRST crossover operator fail; in contrast, when using the slightly modified non-heuristic Kruskal RST* crossover operator, the encoding shows a good performance (similar to NetKeys). When using the heuristic variants of the edge-sets (heur. ini, heur. xover, and heur. ini & xover), the results indicate a complete failure if the optimal solution is a star, a list, or an arbitrary tree even for small problem instances. Due to the strong bias of the heuristic edge-sets towards the MST, GAs using the heuristic variants of the encoding fail when searching for non-MST-like optimal solutions. If the optimal solution is the MST, edge-sets with heuristics allow high GA performance and find the MST very quickly.

We see that the empirical results confirm the theoretical predictions. It is more difficult to predict the performance of direct representation as the properties of problem-specific search operators (like a possible bias) are more difficult to identify. However, the theoretical investigations into the edge-sets (compare Sect. 7.2) illustrated that analyzing the bias of the initialization, crossover, and mutation operators also allow us to make accurate predictions regarding GA performance.

The results show that only NetKeys and the LNB encoding with a large link-specific bias P_1 show high performance for the one-max tree problem independently of the structure of the optimal solution. GAs using the non-heuristic KruskalRST* encoding perform slightly worse but the performance is still high. For the other types of encodings, we confirmed the theoretical predictions and found that GAs using CVs fail for large problem instances due to problems with non-synonymous redundancy, and GAs using Prüfer number fail due to the low locality of the encoding. GAs using the NB encoding or NetDir encoding only perform well if the optimal solution is a star, and GAs using heuristic variants of the edge-set encoding fail if the optimal solution is not the MST.

8.1.3 Deceptive Trap Problem for Trees

We compare the performance of GAs using different representations for the deceptive trap problem for trees.

Problem Definition

The deceptive trap problem for trees is defined in Sect. 6.1.5. The problem is fully difficult for crossover-based GEAs as all tree schemata with $k < n-1$ that contain the optimal solution T_{opt} have lower fitness than their competitors.

For the deceptive tree problem, an optimal solution T_{opt} is chosen a priori either randomly or by hand. As for the one-max tree problem, we assume a minimization problem and the fitness of T_{opt} is $f_{T_{opt}} = 0$. The fitness f_{T_i} of

other individuals $T_i \neq T_{opt}$ is defined as $d_i = n - d_{T_{opt},T_i}$. Therefore, we get for the fitness f_i of an individual T_i

$$f_i = \begin{cases} 0 & \text{if } T_i = T_{opt} \\ n - d_{T_{opt},T_i} & \text{if } T_i \neq T_{opt} \end{cases} \tag{8.1}$$

The lower f_i of an individual T_i, the less links it has in common with the optimal solution. The size of a deceptive trap problem denotes the number of nodes n.

Even small instances of the trap problem for trees can not be solved by GEAs with reasonable population sizes. Therefore, to be able to construct larger problem instances, we concatenate m traps of size n. The size n of a deceptive trap is denoted by the number of nodes that are part of this sub-problem. As a result, by concatenating m deceptive trap problems for trees of size n we can construct problems with overall $m \times n$ nodes. The fitness of an individual is the sum of the fitness of the deceptive sub-problems and is calculated as $\sum_{i=0}^{m-1} f_i$, where f_i is the fitness of a sub-tree (8.1). When concatenating deceptive traps to larger problems, we have to consider that the sub-problems of size n containing $n-1$ links must be connected to form a fully connected tree with $mn-1$ links. Therefore, at least $(mn-1)-m(n-1) = m-1$ links are not considered for the fitness of an individual.

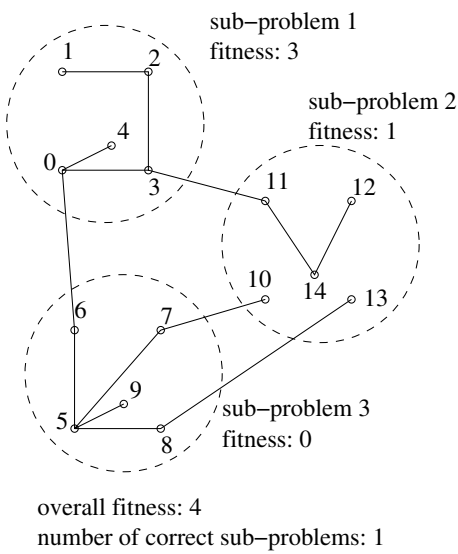


Figure 8.1. Calculation of the fitness for three concatenated, deceptive traps of size $n = 5$. Although a tree with 15 nodes has 14 links, only a maximum of 12 links is considered for the calculation of the fitness.

We illustrate in Fig. 8.1 how the fitness f_i of an individual T_i is calculated if we concatenate three deceptive sub-problems of size $n = 5$. The optimal solution should be the star with center $5i$, where $i \in \{0, 1, 2\}$. If we have three concatenated sub-problems, there exist three groups of nodes each consist of

nodes. For example, sub-problem 3 consists of the nodes 5, 6, 7, 8, and 9. Sub-problem 3 in Fig. 8.1 shows the optimal solution to the sub-problem which is a star with center 5. The fitness of an individual is the sum of the fitness of the deceptive sub-problems. Therefore, the minimum fitness is zero (all three sub-problems are correct and $T_i = T_{opt}$) and the maximum fitness is 12 (each solution of a sub-problem has only one link in common with the optimal solutions of the sub-problem). In our example, sub-problem 2 has no links in common with the optimal solution of the sub-problem and its contribution to the overall fitness is $f_2 = 1$. For more information regarding the deceptive tree problem, the reader is referred to Sect. 6.1.5.

Theoretical Predictions

We predict the performance of GEAs for deceptive trap problems for trees based on the redundancy and locality of the different representations.

Due to uniform redundancy (compare Sect. 8.1.1), we expect the performance of GEAs using uniformly redundant encodings such as CVs, NetKeys, NetDir, LB ($P_1 = 20$), and edge-sets without heuristics to be independent of the structure of the optimal solution.

We have seen in Sect. 8.1.1 that the non-uniformly redundant LNB encoding using a small link-specific or node-specific bias is either biased towards stars or the MST. Therefore, GEA performance would be high if we only had one deceptive sub-problem, and the optimal solution is a star or the MST. However, when concatenating m sub-problems to a larger problem, the overall optimal solution is different from either a star or the MST. Thus, GEAs using LNB encodings with small biases fail for concatenated traps as the optimal solution is neither a star nor the MST. The situation is expected to be similar for edge-sets with heuristics. As these encodings have a bias towards the MST, GEAs using these encodings also fail in finding the optimal solution as the overall optimal solution is not similar to the MST (even if the optimal solutions to the sub-problems are MSTs).

We have learned that low-locality encodings make fully difficult problems easier and fully easy problems more difficult (compare Sect. 3.3.4). Therefore, GEAs using Prüfer numbers will perform well when used on the fully deceptive tree problem where the optimal solution is a non-star. In comparison to the one-max tree problem where GEAs using Prüfer numbers failed completely, we expect a better performance for the trap problems. The low locality of the encoding destroys the deceptive character of the problem and makes it easier to solve for GEAs. Furthermore, because Prüfer numbers have higher locality around stars, GEAs have more difficulty in finding optimal stars than arbitrary trees when solving the trap. Consequently, GEA performance will be higher if the optimal solution of the deceptive trap is a non-star.

Finally, for the non-synonymous redundant CV encoding, we expect the same effects as for the one-max tree problem. Due to the non-synonymity of the encoding, we expect GEAs to need more generations but to correctly

solve a higher number of problems. Although the overall number of fitness evaluations that are necessary to find the optimum strongly increases due to stealth mutation (populations do not converge and the GEA runs until it is stopped), CVs are an efficient encoding for solving deceptive traps as the non-synonymity of the encoding makes the problem easier to solve for GEAs.

Experimental Results

For our experiments we concatenate four instances of a deceptive trap problem for trees of size 3 and 4. The size of a sub-problem denotes the number of nodes. Therefore, the problems for the empirical investigation have either $n \times m = 12$ (size 3) or $m \times n = 16$ nodes (size 4). The minimum fitness of an individual is 0 and the maximum fitness is either $4 \times 2 = 8$ (4 instances of the 3 node trap) or $4 \times 3 = 12$ (four instances of the 4 node trap).

In our experiments we use a simple genetic algorithm without mutation. For the size 3 problems we use a population size of $N = 100$ and for the size 4 problems we use $N = 1,200$. Trap problems of size 4 are already difficult to solve for GAs and larger problems could hardly be solved using standard GAs. Because uniform crossover would result in high BB-disruption we use two-point crossover in all runs. Furthermore, we use tournament selection without replacement of size 3. The runs are stopped after the population is fully converged or the number of generations exceeds 200 generations. We generate 200 random problems for each problem instance and perform 25 runs for the different representations (compare Sect. 8.1.1) and for each of the randomly generated problems. The optimal solutions for the sub-problems are either an arbitrary tree, the MST, or a star. Because we have seen for the one-max tree problem that GAs show the same performance when searching for lists as when searching for arbitrary trees, we neglect the case that the optimal solution is a list. As before, the GA performance is determined by the percentage P_{succ} of correctly solved problems at the end of the run and the number of generations t_{conv} until the population is fully converged or the GA run is stopped.

The performance of GAs using different representations for size 3 and 4 trap problems is shown in Table 8.5. We present results for different optimal solutions (T_{opt} is either an arbitrary tree, the MST, or a random star) and show the percentage P_{succ} of runs that find the optimal solution T_{opt}, the mean and standard deviation of the fitness of the best found solution, and the mean and standard deviation of the number of generations t_{conv} until the population is fully converged or the GA run is stopped after 200 generations.

As predicted, the performance of unbiased and uniformly redundant encodings (CV, NetKey, NetDir and LNB ($P_1 = 20$)) is nearly independent of the structure of the optimal solution. Because LNB encodings with small biases P_1 and P_2 are biased either towards stars or the MST, the performance of GAs using these types of encodings is low. Because the optimal solutions have nothing in common with a star or the MST, GAs fail. The situation

Table 8.5. Performance of GAs using different types of representations for deceptive tree problems of different sizes and with different T_{opt} (arbitrary tree, MST, and star)

T_{opt}		oder 3					order 4				
		P_{succ}	fitness		t_{conv}		P_{succ}	fitness		t_{conv}	
			μ	(σ)	μ	(σ)		μ	(σ)	μ	(σ)
arbitrary tree	Prüfer	0.54	0.49	(0.6)	26.0	(8.6)	0.17	1.09	(0.6)	100.0	(44.3)
	NetKey	0.78	0.23	(0.4)	23.0	(6.0)	0.15	1.37	(0.8)	95.3	(30.0)
	CV	1	0	(0)	199.9	(0.4)	0.86	0.14	(0.3)	200	(0)
	NB (P_2=1)	0	2.58	(0.3)	15.8	(5.8)	0	3.12	(0.4)	62.6	(29.2)
	NB (P_2=20)	0	2.62	(0.2)	19.7	(6.6)	0	3.06	(0.4)	53.2	(35.5)
	LB (P_1=1)	0.09	1.69	(0.8)	19.5	(7.6)	0	3.00	(0.7)	96.4	(34.0)
	LB (P_1=20)	0.82	0.18	(0.4)	23.3	(6.1)	0.16	1.35	(0.9)	96.8	(30.4)
	P_1=P_2=1	0.12	1.24	(0.6)	27.4	(8.0)	0.01	2.19	(0.7)	130.5	(32.0)
	NetDir	0.94	0.06	(0.2)	107.0	(52.4)	0.54	0.65	(0.8)	162.2	(40.1)
	KrukalRST	0.93	0.07	(0.2)	141.2	(63.9)	0	4.00	(0)	111.3	(37.5)
	KruskalRST*	0.84	0.16	(0.4)	12.2	(1.8)	0	4.00	(0)	37.6	(7.5)
	heur. ini	0	3.83	(0.1)	0.4	(0.3)	0	4.04	(0.1)	1.4	(1.6)
	heur. xover	0	2.63	(0.5)	8.7	(2.4)	0	3.98	(0)	11.2	(3.1)
	h. ini & xover	0	3.88	(0.1)	0.4	(0.4)	0	4.07	(0.1)	0.6	(0.8)
MST	Prüfer	0.49	0.54	(0.5)	26.4	(8.6)	0.15	1.14	(0.6)	104.1	(43.5)
	NetKey	0.78	0.23	(0.4)	23.3	(6.1)	0.14	1.41	(0.8)	96.5	(30.6)
	CV	1	0	(0)	199.9	(0.4)	0.88	0.12	(0.3)	200	(0)
	NB (P_2=1)	0	2.53	(0.2)	16.3	(5.2)	0	2.96	(0.1)	54.5	(14.9)
	NB (P_2=20)	0	2.61	(0.2)	19.4	(6.3)	0	2.98	(0.1)	45.1	(21.6)
	LB (P_1=1)	0.14	1.41	(0.6)	20.9	(6.8)	0	2.52	(0.5)	104.9	(29.5)
	LB (P_1=20)	0.84	0.17	(0.4)	23.5	(6.1)	0.16	1.33	(0.8)	96.0	(30.7)
	P_1=P_2=1	0.15	1.12	(0.5)	27.9	(7.9)	0.02	1.97	(0.5)	127.7	(29.8)
	NetDir	0.91	0.09	(0.3)	108.2	(53.2)	0.47	0.77	(0.8)	158.6	(40.1)
	KrukalRST	0.93	0.07	(0.2)	143.5	(63.9)	0	4.00	(0)	110.0	(36.3)
	KruskalRST*	0.83	0.18	(0.4)	12.2	(1.9)	0	4.00	(0)	37.3	(7.5)
	heur. ini	0	3.81	(0)	0.4	(0.1)	0	4.04	(0)	3.1	(0.8)
	heur. xover	0	2.48	(0.4)	9.1	(2.5)	0	3.96	(0)	13.0	(1.3)
	h. ini & xover	0	3.85	(0)	0.3	(0.1)	0	4.12	(0)	1.3	(0.2)
random star	Prüfer	0.50	0.54	(0.6)	26.4	(8.6)	0.07	1.43	(0.7)	111.2	(46.8)
	NetKey	0.78	0.22	(0.4)	23.1	(5.9)	0.18	1.28	(0.8)	94.7	(30.1)
	CV	1	0	(0)	200	(0.2)	0.91	0.09	(0.3)	200	(0)
	NB (P_2=1)	0	2.58	(0.3)	15.6	(5.6)	0	2.84	(0.1)	48.4	(14.2)
	NB (P_2=20)	0	2.59	(0.3)	19.3	(6.5)	0	2.71	(0.2)	76.5	(24.6)
	LB (P_1=1)	0.08	1.71	(0.8)	19.3	(7.6)	0	2.90	(0.7)	94.1	(35.1)
	LB (P_1=20)	0.81	0.20	(0.4)	23.5	(6.2)	0.20	1.21	(0.8)	95.3	(30.3)
	P_1=P_2=1	0.10	1.28	(0.6)	27.3	(8.1)	0.03	1.70	(0.6)	131.6	(30.7)
	NetDir	0.90	0.10	(0.3)	108.0	(52.9)	0.57	0.58	(0.8)	160.2	(40.8)
	KrukalRST	0.93	0.07	(0.2)	143.2	(63.6)	0	4.00	(0)	120.8	(38.6)
	KruskalRST*	0.84	0.17	(0.4)	12.2	(1.9)	0	4.00	(0)	39.2	(7.6)
	heur. ini	0	3.85	(0.1)	0.4	(0.3)	0	4.04	(0.1)	1.3	(1.5)
	heur. xover	0	2.62	(0.5)	8.7	(2.5)	0	3.98	(0)	11.2	(2.7)
	h. ini & xover	0	3.89	(0.1)	0.3	(0.3)	0	4.05	(0.1)	0.6	(0.7)

is similar when using the heuristic variants of the edge-set encoding. Due to their strong bias towards the MST, GAs fail for deceptive traps independently of the optimal solution of the sub-problems, as the overall optimal solution is not similar to the MST. The non-heuristic variants of the edge-sets show good performance for the smaller problem instances but fail for the larger problems.

In comparison to the fully easy one-max problems, GAs using Prüfer numbers perform well. The low locality of the encoding helps GAs find their way to the optimal solution. As expected, the solution quality is lower when searching for optimal stars (in comparison to arbitrary trees or the MST) because the locality around star-like structures is higher.

As we have already seen for the one-max tree problem, the LB encoding with a large link-specific bias $P_1 = 20$ results in the same GA performance as when using the NetKey encoding. Although both representations are synonymously redundant, GAs using one of these two encodings are able to solve the problem and only need a few generations. Due to the non-synonymity of CVs, GAs using the CV encoding perform very well for the difficult deceptive tree problems. However, the non-synonymity of the encoding increases the number of generations. The results for the NetDir representation are surprising. GAs using this encoding are able to find a high proportion of BBs but need a larger number of generations.

Coincidentally with the results for the one-max tree problem, NetKeys and LNB encodings with a large link-specific bias allow GEAs to reliably solve concatenated trap problems for trees after a few generations. In contrast to the one-max tree problem, GEAs using Prüfer numbers or CVs show a comparable or higher performance for this difficult problem as the low locality of Prüfer numbers and the non-synonymity of CVs make difficult problems easier to solve.

8.2 GEA Performance on Optimal Communication Spanning Tree Problems

This section illustrates how we can use the framework about representations for predicting and verifying differences of GEA performance for some instances of the optimal communication spanning tree (OCST) problem. Consequently, this section also provides a comprehensive comparison for the performance of different tree representations on OCST problems. We present results for test instances from Palmer (1994), Raidl (2001), Berry et al. (1995), and some new test problems.

In Sect. 8.2.1, we give a brief description of the OCST problem. Section 8.2.2 gives an overview over different approaches from the literature that have been used for solving the OCST problem. This is followed in Sect. 8.2.3 by a short description of the test problems (further details can be found in Appendix A). Then, in the remaining subsections, the influence of representations on

GA performance is studied. Based on an analysis of the properties of representations, which is performed in Sect. 8.2.4, Sect. 8.2.5 gives some predictions on the expected GA performance for the test instances. The predictions and the validity of the developed theoretical concepts for the OCST test instances are verified in Sect. 8.2.6.

8.2.1 The Optimal Communication Spanning Tree Problem

The OCST problem (also known as minimum communication spanning tree problem or simple network design problem (Johnson et al. 1978)) was introduced in Hu (1974). The problem is listed as [ND7] in Garey and Johnson (1979) and Crescenzi and Kann (2003). For the OCST problem, the number and positions of network nodes are given a priori and the cost of the tree is determined by the cost of the links. A link's flow is the sum of the communication demands between all pairs of nodes communicating either directly, or indirectly, over the link. The goal is to find a tree that connects all given nodes and satisfies their communication requirements for a minimum total cost. The cost for each link is not fixed a priori but depends on its distance weight and its capacity. A link's capacity must satisfy the flow over this link, which depends on the entire tree structure.

The OCST problem can formally be defined as follows. An undirected graph is denoted as $G = (V, E)$. $n = |V|$ denotes the number of nodes and $|E|$ denotes the number of edges. There are communication or transportation demands between the n different nodes. The demands are specified by an $n \times n$ demand matrix $R = (r_{ij})$, where r_{ij} is the amount of traffic required between location v_i and v_j. An $n \times n$ distance matrix $D = d_{ij}$ determines the distance weights associated with each pair of sites. A tree $T = (V, F)$ where $F \subseteq E$ and $|F| = |V| - 1$ is called a *spanning tree* of G if it connects all the nodes. The weight $c(T)$ of the spanning tree is the weighted sum over all pairs of vertices of the cost of the path between all pairs in T. In general,

$$c(T) = \sum_{i,j \in F} f\left(d_{ij}, b_{ij}\right),$$

where the $n \times n$ matrix $B = b_{ij}$ denotes the traffic flowing directly and indirectly over the edge between the nodes i and j. It is calculated according to the demand matrix R and the structure of T. T is the minimum communication spanning tree if $c(T) \leq c(T')$ for all other spanning trees T'.

For the OCST problem as proposed by Hu (1974) the cost of a link is calculated as the product of the distance weight d_{ij} times the overall traffic b_{ij} running over the edge. Therefore, $f = d_{ij} b_{ij}$ and

$$c(T) = \sum_{i,j \in V,\ i<j} r_{ij} \times d(p_{i,j}^T), \tag{8.2}$$

where $d(p_{i,j}^T)$ denotes the weight of the unique path from node i to node j in the spanning tree T. The OCST problem seeks the spanning tree with minimal costs among all other spanning trees.

The OCST problem becomes the minimum spanning tree (MST) problem if $f = d_{ij}$. No communication requirements r_{ij} are considered. Then, T is the minimum spanning tree if $c(T) \leq c(T')$ for all other spanning trees T', where

$$c(T) = \sum_{(i,j)\in E} d_{ij} \tag{8.3}$$

8.2.2 Optimization Methods for the Optimal Communication Spanning Tree Problem

Like other constrained spanning tree problems, the OCST problem is $\mathcal{NP}$-hard (Garey and Johnson 1979, p. 207). Furthermore, it was shown in Reshef (1999) that the problem is $\mathcal{MAX}$ $\mathcal{SNP}$-hard (Papadimitriou and Yannakakis 1991) which means it cannot be solved using a polynomial-time approximation scheme, unless $\mathcal{P} = \mathcal{NP}$. Therefore, the OCST problem belongs to the class of optimization problems that behave like MAX-3SAT (Garey and Johnson 1979).

Only for a few easy and restricted problem instances have algorithms been developed that return optimal solutions. Hu (1974), who introduced the OCST problem, gave exact algorithms for two specific versions of the OCST problem. He showed that for the *complete unweighted graph* version, where $d_{ij} = 1$ for every i and j, the problem can be solved in polynomial time using the Gomory-Hu spanning tree algorithm (Gomory and Hu 1961; Hu 1974). Hu called this the *optimum requirement spanning tree problem*. In addition, he showed for the *uniform demand* version of the OCST, where the communication demands r_{ij} between any two sites are equal, that the optimal solution is a star if the distance weights d_{ij} satisfy a stronger version of the triangle inequality: for every $1 \leq i, j, k \leq n$ such that $d_{ij} \leq d_{ik} \leq d_{jk}$, we have $(d_{jk} - d_{ij})/d_{ik} \leq (n-2)/(2n-2)$. If both the communication demands r_{ij} and the distances weights d_{ij} between any two sites are equal, then the optimal solution is also a star. Later, Johnson et al. (1978) showed that only the uniform demand version, where the d_{ij} satisfy this stronger version of the triangle equation, can be solved in polynomial time, and that all other uniform demand versions of the OCST problem, where $d_{ij} \in \{1, \infty\}$ are $\mathcal{NP}$-hard. Wu et al. (1998) extended this work and showed that the uniform demand version where the weights d_{ij} satisfy the triangle inequality is $\mathcal{NP}$-hard. For the uniform demand version, Wong (1980) presented a heuristic that finds a tree T which has a maximum cost which is twice that of the optimal solution, $c(T) \leq 2c(T_{opt})$.

The development of exact optimization methods for the general, non-uniform demand version of the OCST problem showed less success. Some early work (Dionne and Florian 1979; Lin 1982; Gavish 1983; Gavish and

Altinkemer 1990) addressed the general network design problem and developed heuristics for finding optimal graphs G (not trees T) for given distance weights d_{ij} and demands r_{ij}. However, as the assumptions that are made for solving the general network design problem are incompatible with solving the OCST problem (Palmer 1994, p. 10ff), these heuristics can not be applied to the OCST problem. Later, Peleg (1997) showed that the OCST problem is reducible to a problem called *minimum average stretch spanning tree* (MAST) problem. Therefore, both problems are equivalent to each other and approximation algorithms for the MAST problem can also be used for the OCST problem. In the MAST problem, which was introduced in Alon et al. (1995), a graph G and a distance matrix D is given, and a spanning tree T has to be found that minimizes the average stretch of the edges (e.g. minimize $\frac{1}{n-1}\sum_{i,j\in E} d(p_{i,j}^T)/d_{ij}$, where $d(p_{i,j}^T)$ is the sum of all the weights along the path between i and j in the spanning tree T). Alon et al. presented a randomized algorithm for the MAST problem that constructs a spanning tree such that the average cost of the tree is less than, or equal to, $\exp(O(\sqrt{\log n \log\log n}))$.

Other approximation algorithms for the OCST problem are based on the volume of communication $c(G) = \sum_{i,j\in E} r_{ij} d(p_{i,j}^G)$ in the complete graph G, where $d(p_{ij}^G)$ is the sum of all the weights along the shortest path between i and j in G. $c(G)$ represents a trivial lower bound for $c(T)$ because it considers the full original graph G and not only the links used for the tree T. Bartal (1996) and Wu et al. (1998) presented a randomized algorithm that constructs a spanning tree T with expected communication cost $c(T) = O(\log^2 n)c(G)$. This result has been improved by Bartal (1998) to an $O(\log n \log\log n)$ approximation. Around the same time, non-randomized, deterministic algorithms were developed that find a spanning tree with cost $c(T) = O(\log^2 n)c(G)$ (Peleg and Reshef 1998; Reshef 1999). Charikar et al. (1998) improved these results and presented a deterministic approximation algorithm that results in $c(T) = O(\log n \log\log n)c(G)$. When using Euclidean distances as distance weights d_{ij}, Charikar et al. (1998) and Reshef (1999) presented deterministic approximation algorithms that output a spanning tree with cost $c(T) = O(\log n)c(G)$. Despite the progress in developing approximation algorithms for the OCST problem that are based on the volume of communication $c(G)$, Alon et al. (1995) showed that such approximation techniques cannot approximate the OCST problem better than $\Omega(\log n)$. For more detailed information about approximation algorithms for the OCST problem, we refer to Reshef (1999).

When summarizing the development of optimization algorithms for the OCST problem, we conclude that no efficient algorithmic methods for solving the OCST problem are available. Some algorithms exist for simplified versions of the OCST problems (complete unweighted graph problem and uniform demand problems), but there are no efficient methods for standard OCST problems. Similarly, deterministic and randomized approximation algorithms for the OCST problem are available which are based on the volume of commu-

nication $c(G)$, but none of them are able to output optimal or near-optimal solutions ($c(T) \geq \Omega(\log n)c(G)$).

To overcome the limitations of exact and approximation algorithms, and to be able to find optimal or near-optimal solutions for OCST problems, researchers have used heuristic optimization methods like GEAs, simulated annealing, tabu search, and other approaches. One of the first heuristic approaches for the OCST problem was presented by Palmer (1994). When applying GEAs to the OCST problem, he recognized that the design of a proper tree representation is crucial for the performance of GEAs. Other representations that are used for the OCST problem or similar problems are the CV encoding (Berry et al. 1994; Berry et al. 1995), weighted encodings like the LNB encoding (Palmer 1994), the weighted encoding (Raidl and Julstrom 2000), the NetKey encoding (Rothlauf et al. 2002), different variants of the LNB-encoding (Krishnamoorthy and Ernst 2001), direct representations like edge-sets (Raidl and Julstrom 2003; Tzschoppe et al. 2004) or other direct representations (Li 2001), determinant factorization (Abuali et al. 1995), Prüfer numbers (Palmer 1994; Palmer and Kershenbaum 1994a; Kim and Gen 1999; Zhou and Gen 1997; Krishnamoorthy et al. 1999; Gen et al. 1998; Gen et al. 1998; Gargano et al. 1998; Rothlauf and Goldberg 1999; Gottlieb et al. 2001; Julstrom 2001), or variants of Prüfer numbers like the Blob Code, the Happy Code and the Dandelion Code (Picciotto 1999; Julstrom 2001; Julstrom 2005). Other work applying GEAs to OCST and related problems (e.g. degree constraint MST problems) have been presented by Premkumar et al. (2001) and Chu et al. (2000)).

It was shown in Rothlauf et al. (2003) that on average, optimal solutions for OCST problems are similar to the MST (compare also Sect. 7.2.3). That means the average distance $d_{opt,MST}$ between the optimal solution and the MST is significantly lower than the average distance $d_{rnd,MST}$ between a randomly created tree and the MST (Fig. 7.6). Therefore, as the optimal solution of an OCST problem is biased towards the MST, representations as well as operators that favor or overrepresent trees that are similar to the MST are expected to solve the OCST problem more efficiently.

In summary, due to the lack of efficient algorithmic methods for finding optimal or near-optimal solutions, a large amount of work has applied GEAs to OCST problems. When using GEAs for tree problems, the choice of a proper tree representation is one of the most important factors for success.

8.2.3 Description of Test Problems

Several instances of the OCST problem have been presented in the literature.

The oldest test instances are from Palmer who introduced OCST test problems with 6, 12, 24, 47, and 98 nodes (Palmer 1994). The demands r_{ij} were inversely proportional to the distance weights d_{ij}. The nodes correspond to cities in the United States, and the distance weights d_{ij} were obtained from a tariff database. In analogy to (8.2) the cost $c(T)$ of a tree is defined as

$$c(T) = \sum_{i,j \in V,\, i<j} r_{ij} \times d(p_{i,j}^T), \tag{8.4}$$

where $d(p_{i,j}^T)$ denotes the weight of the unique path from node i to node j in the spanning tree T. For the exact distance and requirement matrix for the 6, 12 and 24 node problem the reader is referred to Palmer (1994) or Appendix A.1. Unfortunately, the data for the 47 and 98 node problems is no longer available[1].

Raidl (2001) proposed several test instances of the OCST problem ranging from 10 to 100 nodes. The distances weights d_{ij} and the traffic demands r_{ij} have been generated randomly. The cost of a tree is calculated in analogy to Palmer's test instances (compare (8.2) or (8.4)) The distance matrix and traffic demands are summarized in Appendix A.2.

Berry et al. (1995) presented three instances of the OCST problem. They proposed one 6 node and two 35 node problems. The distance weights d_{ij} and the traffic demands r_{ij} are listed in Appendix A.3. As before, the cost of a tree is calculated according to (8.2) or (8.4)). Both 35 node problems use the same traffic demands r_{ij}, but differ in the distance weights. One problem has uniform weights with $d_{ij} = 1$ (berry35u), and the other has non-uniform weights (berry 35). The best solution found by Berry et al. (1995) for the problem berry35 has cost $c(T_{opt}) = 30,467$. Li and Bouchebaba (1999) improved these result to 16,915. The problem berry35u is an optimum requirement spanning tree problem and can be solved in polynomial time using the Gomory-Hu spanning tree algorithm (compare Sect. 8.2.2). The optimal solution has cost 16,273.

The fourth group of test problems consists of problems that are derived from a real-world 26-node problem from a company with locations all over Germany. For fulfilling the demands between the nodes, different line types with only discrete capacities are available. The cost for installing a link consists of a fixed share and a share which depends on its distance weight. Both cost components depend on the capacity of the link. The cost are based on the tariffs of the German Telecom from 1996. For an exact description on how the cost of a link depends on its weight d_{ij} and its capacity b_{ij}, the reader is referred to Appendix A.4.

There are four different problems. Rothlauf1 is a problem with $n = 16$ nodes and traffic demands ending only at node v_1 ($r_{ij} = 0$, for $i, j \neq 1$). In rothlauf2, there are only $n = 15$ nodes and some additional traffic is added. Rothlauf3 is similar to rothlauf1 but uses a modified cost function. Finally, in rothlauf4 with $n = 16$, there is traffic between all nodes. The distance weights d_{ij} between the nodes are calculated as the Euclidean distances. The known best solutions for the four test problems are shown in Fig. A.1. We summarize the most important properties of the four test problems:

[1]The data sets were not listed in the thesis and are not directly available from Palmer.

- **rothlauf1: One headquarter and 15 branch offices**: This problem is the original design problem. All 15 branch offices (node 2 to 16) communicate only with the headquarter (node 1). Possible line capacities are 64 kBit/s, 512 kBit/s, and 2048 kBit/s.
- **rothlauf2: One headquarter and only 14 branches**: In this problem, one node is removed from the graph and some additional traffic is added.
- **rothlauf3: One headquarter, 15 branches and cheap lines for everybody**: In this scenario, the fixed cost for installing a line is only 10% of the cost in rothlauf1. Therefore, the cost of a link is mainly determined by its distance weight. Hence, the optimal solution is more like a minimum spanning tree. If the link costs would only be determined by the distance weights, and if there was only one possible capacity, the optimal solution would be the MST (compare (8.3)).
- **rothlauf4: 4 headquarters, 12 branches and all working together**: For this problem, the demand matrix is completely filled. Between every node i and j some traffic exists. Between the four headquarters (nodes 1, 2, 3 and 4), the traffic is randomly chosen between 256 kBit/s and 512 kBit/s. Every other node communicates with the four headquarters and has a randomly chosen demand between 0 and 512 kBit/s. This demand is split into the headquarters at a ratio of 0.4, 0.3, 0.2, and 0.1 for the nodes 1, 2, 3, and 4[2]. Between all 12 branch offices the demand is randomly chosen between 0 and 64 kBit/s. To make the problem more realistic, two additional line types are available. It is possible to use a line with 128 kBit/s and 4,096 kBit/s capacity with twice the cost of a 64kBit/s and 2,048 kBit/s line.

8.2.4 Analysis of Representations

As we have seen in Sect. 8.1.1, an analysis of the bias of representations is important for evaluating the influence of representations on GEA performance and to be able to make accurate predictions about the expected GEA performance. Therefore, we perform the same type of analysis as described in Sect. 8.1.1. For each representation, we randomly generate 10,000 genotypes and measure the mean μ and the standard deviation σ of the minimum phenotypic distance towards a star, $\min(d^p_{rnd,star})$, and the phenotypic distance towards the MST, $d^p_{rnd,MST}$. For the different problems, we use the distance weights from the Appendix A. When using edge-sets, the number N of individuals that are generated influence the structure of the encoded tree (compare Sect. 7.2.1). Therefore, for edge-sets, we created 100 random populations of size $N = 100$ resulting in 10,000 random individuals.

[2]Node 1 is the most important node and 40% of the traffic of the branches ends there; in node 2, 30% of the traffic ends, and so on.

Table 8.6. We generate random genotypes x^g_{rnd} for the test instances from Palmer and Raidl and calculate the minimum phenotypic distance towards star networks, $\min(d^p_{rnd,star})$, and the phenotypic distance to the MST, $d^p_{rnd,MST}$. The results confirm the results presented in Sect. 8.1.1.

	problem	palmer6	palmer12	palmer24	raidl10	raidl20
	distance d	μ (σ)	μ (σ)	μ (σ)	μ (σ)	μ (σ)
Prüfer number	$\min(d^p_{rnd,star})$	2.04 (0.6)	7.22 (0.7)	18.50 (0.8)	5.42 (0.7)	14.69 (0.7)
	$d^p_{rnd,MST}$	3.36 (0.9)	9.17 (1.2)	21.05 (1.3)	7.20 (1.1)	17.07 (1.3)
NetKey	$\min(d^p_{rnd,star})$	2.12 (0.6)	7.30 (0.7)	18.61 (0.8)	5.51 (0.7)	14.78 (0.8)
	$d^p_{rnd,MST}$	3.34 (0.9)	9.15 (1.2)	21.07 (1.3)	7.21 (0.8)	17.11 (0.8)
CV	$\min(d^p_{rnd,star})$	2.04 (0.6)	7.22 (0.7)	18.50 (0.8)	5.41 (0.7)	14.67 (0.8)
	$d^p_{rnd,MST}$	3.38 (0.9)	9.16 (1.2)	21.04 (1.3)	7.23 (1.1)	17.05 (1.3)
NB (P_2=1)	$\min(d^p_{rnd,star})$	1.00 (0.8)	5.49 (1.3)	7.59 (4.2)	4.12 (1.1)	10.75 (2.2)
	$d^p_{rnd,MST}$	2.37 (0.8)	6.12 (1.1)	18.68 (1.6)	4.24 (1.1)	11.63 (1.3)
NB (P_2=20)	$\min(d^p_{rnd,star})$	0.08 (0.3)	0.64 (1.4)	0.71 (2.1)	0.59 (1.1)	2.62 (3.2)
	$d^p_{rnd,MST}$	3.26 (0.5)	8.86 (0.7)	20.90 (0.9)	6.91 (0.9)	16.34 (1.4)
LB (P_1=1)	$\min(d^p_{rnd,star})$	2.08 (0.6)	7.50 (0.6)	18.61 (0.8)	5.41 (0.7)	14.78 (0.8)
	$d^p_{rnd,MST}$	2.50 (1.0)	6.55 (1.4)	19.45 (1.6)	4.61 (1.3)	12.74 (1.8)
LB (P_1=20)	$\min(d^p_{rnd,star})$	2.13 (0.6)	7.30 (0.7)	18.59 (0.8)	5.51 (0.7)	14.78 (0.8)
	$d^p_{rnd,MST}$	3.28 (0.9)	9.0 (1.2)	21.01 (1.3)	7.02 (1.1)	16.77 (1.3)
LNB (P_1=P_2=1)	$\min(d^p_{rnd,star})$	1.59 (0.7)	6.22 (1.1)	15.39 (1.7)	4.66 (1.0)	12.48 (1.5)
	$d^p_{rnd,MST}$	2.68 (0.9)	7.12 (1.3)	20.21 (1.4)	5.11 (1.2)	14.02 (1.7)
NetDir	$\min(d^p_{rnd,star})$	2.05 (0.6)	7.22 (0.7)	18.50 (0.8)	5.42 (0.7)	14.68 (0.8)
	$d^p_{rnd,MST}$	3.39 (0.9)	9.19 (1.2)	21.05 (1.3)	7.22 (1.1)	17.04 (1.3)
KruskalRST, KruskalRST*, heur xover	$\min(d^p_{rnd,star})$	2.12 (0.6)	7.30 (0.7)	18.60 (0.8)	5.50 (0.7)	14.78 (0.7)
	$d^p_{rnd,MST}$	3.30 (0.9)	9.17 (1.1)	21.07 (1.3)	7.20 (1.1)	17.11 (1.2)
heur ini, heur ini & xover	$\min(d^p_{rnd,star})$	2.86 (0.3)	8.64 (0.5)	19.79 (0.41)	5.90 (0.3)	15.0 (0.1)
	$d^p_{rnd,MST}$	0.31 (0.6)	0.66 (0.8)	1.72 (1.8)	0.15 (0.3)	2.22 (0.6)

In Tables 8.6 (test instances from Palmer (1994) and Raidl (2001)) and 8.7 (test instances from Berry et al. (1995) and real-world test instances) we show the results for the different representations.

The results confirm the findings from Sect. 8.1.1. The use of uniformly redundant encodings (NetKeys, CVs, or LB ($P_1 = 20$)) results in about the same distances as when unbiased and non-redundant Prüfer numbers are used. The situation is the same for unbiased direct encodings (NetDir and edge-sets with non-heuristic initialization).

The situation is different for non-uniform redundant or biased encodings. The two variants of the NB encoding, which are non-uniformly redundant, show a bias towards stars. As expected, the bias towards stars increases with increasing P_2 (compare Sect. 6.4.3). For the LB encoding, the distances between randomly generated solutions and the MST decreases with lower P_1, as variants of the LB and LNB encoding with a low link-specific bias P_1 overrepresent MST-like trees. Finally, the results for variants of the edge-sets that use heuristic initialization (heur. ini, heur. ini & xover) confirm the strong bias

Table 8.7. We generate random genotypes x^g_{rnd} for the real-world test instances and the test instances from Berry and calculate the minimum phenotypic distance towards star networks, $\min(d^p_{rnd,star})$, and the phenotypic distance to the MST, $d^p_{rnd,MST}$.

	problem	berry6	berry35u	berry35	rothlauf2	rothlauf1, 3, 4
	distance d	μ (σ)	μ (σ)	μ (σ)	μ (σ)	μ (σ)
Prüfer number	$\min(d^p_{rnd,star})$	2.03 (0.6)	29.2 (0.8)	29.2 (0.8)	9.99 (0.8)	10.9 (0.8)
	$d^p_{rnd,MST}$	3.51 (0.8)	n.a.	32.05 (1.3)	12.1 (1.1)	13.1 (1.2)
NetKey	$\min(d^p_{rnd,star})$	2.11 (0.6)	29.3 (0.8)	29.3 (0.8)	10.1 (0.7)	11.0 (0.8)
	$d^p_{rnd,MST}$	3.34 (0.9)	n.a.	32.0 (1.3)	12.1 (1.0)	13.1 (1.2)
CV	$\min(d^p_{rnd,star})$	2.04 (0.6)	29.2 (0.8)	29.2 (0.8)	9.98 (0.8)	10.9 (0.8)
	$d^p_{rnd,MST}$	3.54 (0.8)	n.a.	32.0 (1.3)	12.1 (1.0)	13.1 (1.2)
NB (P_2=1)	$\min(d^p_{rnd,star})$	1.48 (0.6)	0 (0)	21.6 (3.5)	7.6 (1.8)	7.91 (1.8)
	$d^p_{rnd,MST}$	1.78 (0.9)	n.a.	22.1 (2.7)	8.06 (1.4)	8.66 (1.5)
NB (P_2=20)	$\min(d^p_{rnd,star})$	0.16 (0.4)	0 (0)	3.84 (4.7)	0.00 (0)	0.00 (0.1)
	$d^p_{rnd,MST}$	3.20 (0.8)	n.a.	30.91 (1.7)	12.13 (0.6)	13.12 (0.6)
LB (P_1=1)	$\min(d^p_{rnd,star})$	2.15 (0.5)	29.3 (0.8)	28.9 (0.9)	10.0 (0.8)	11.0 (0.8)
	$d^p_{rnd,MST}$	1.86 (0.9)	n.a.	23.9 (2.4)	8.85 (1.6)	9.6 (1.7)
LB (P_1=20)	$\min(d^p_{rnd,star})$	2.11 (0.6)	29.3 (0.8)	29.3 (0.8)	10.1 (0.8)	11.0 (0.8)
	$d^p_{rnd,MST}$	3.25 (0.9)	n.a.	31.4 (1.5)	11.9 (1.3)	12.9 (1.3)
LNB (P_1=P_2=1)	$\min(d^p_{rnd,star})$	1.77 (0.6)	24.6 (1.9)	25.4 (2.0)	8.4 (1.3)	9.20 (1.3)
	$d^p_{rnd,MST}$	2.14 (0.9)	n.a.	26.54 (2.2)	9.75 (1.4)	10.5 1.5)
NetDir	$\min(d^p_{rnd,star})$	2.04 (0.6)	29.2 (0.8)	29.2 (0.8)	9.98 (0.8)	10.91 (0.8)
	$d^p_{rnd,MST}$	3.50 (0.8)	n.a.	32.0 (1.3)	12.1 (1.1)	13.1 (1.2)
KruskalRST, KruskalRST*, heur xover	$\min(d^p_{rnd,star})$	2.12 (0.6)	29.3 (0.8)	29.2 (0.8)	10.1 (0.7)	11.0 (0.7)
	$d^p_{rnd,MST}$	3.34 (0.9)	n.a.	32.0 (1.3)	12.1 (1.2)	13.1 (1.2)
heur ini, heur ini & xover	$\min(d^p_{rnd,star})$	2.0 (0)	23.6 (0.7)	30 (0)	11.0 (0)	12.0 (0)
	$d^p_{rnd,MST}$	0.12 (0.3)	n.a.	0 (0)	0.66 (0.7)	0.45 (0.6)

of these encodings towards the MST. Randomly created solutions are only slightly different from the MST.

The findings for the different representations are consistent for the different test problems. The reader should bear in mind that it is not meaningful to calculate $d^p_{rnd,MST}$ for the problem berry35u as all distance weights $d_{ij} = 1$, for $i, j \in V$. Therefore, no unique MST exists for this problem.

8.2.5 Theoretical Predictions on the Performance of Representations

It is difficult to predict GEA performance on a problem of unknown complexity. If we have no information regarding the structure of the optimal solution and the difficulty of the problem, we are not able to make any predictions about GEA performance. However, we know from experience that many real-world problems are easy. Therefore, the problems are often decomposable and the BBs are not fully deceptive, but of lower order.

Therefore, due to the expected low difficulty of OCST problems (compare Sect. 3.3.4) we expect GEAs using Prüfer numbers to fail when used on these problems. The low locality of the encoding will increase the difficulty of easy problems and the use of GEAs results more in a random than a guided search. Furthermore, we know that the performance of GEAs using either uniformly redundant encodings (NetKeys, NetDir, or LNB encodings with large link-specific bias P_1), or unbiased direct representations (NetDir or the non-heuristic variants of the edge-sets), is independent of the structure of the optimal solution. For the CV encoding, its non-synonymous redundancy results in a higher evolvability and allows GEAs to reach a higher number of different phenotypes. Therefore, we expect for smaller problems a better solution quality but a much higher number of search steps. For larger problems instances, we expect GEA failure due to the non-synonymous redundancy of CVs.

An important problem-specific property of the OCST problem was found by Rothlauf, Gerstacker, and Heinzl (2003) (compare also Sect. 7.2.3). This work performed a statistical analysis on the properties of optimal solutions T_{opt} for randomly generated OCST problems using random demands r_{ij} and either Euclidean (on a two-dimensional grid), or random distance weights d_{ij}. They compared the average distances $\mu(d_{MST,rnd})$ of randomly created trees towards the MST to the average distances $\mu(d_{mst,opt})$ of the optimal solutions towards the MST. The results show that the average distance between the optimal solution T_{opt} and the MST is significantly smaller than the average distance between a randomly created tree and the MST (compare Fig. 7.6). Therefore, optimal solutions for the OCST problem are biased towards the MST.

We have seen in Sects. 8.2.4 and 8.1.1 that the LB and LNB encoding using a small link-specific bias P_1 as well as the edge-set encoding with heuristics are biased towards the MST. When using these types of encodings, the average distance between a randomly created solution and the MST is smaller in comparison to a non-redundant and unbiased encoding. As the results presented in Fig. 7.6 indicate that in comparison to randomly created solutions optimal solutions are more similar to the MST, we expect good GEA performance for variants of the LNB encoding using a low link-specific bias. Such variants of the LB and LNB encoding are biased towards the MST and can make use of this problem-specific property of the OCST problem. Furthermore, Sect. 7.2.3 has shown that the performance of edge-sets with heuristics is low if the optimal solution is not the MST. As for edge-sets with heuristics the bias towards the MST is too strong, problems can only be solved where the optimal solution is the MST. Therefore, we expect difficulties when using edge-sets with heuristics for OCST problems as only problems can be solved where the optimal solution is slightly different from the MST.

In analogy, we expect low performance for variants of the NB encoding, as these encodings overrepresent trees that are similar to stars.

8.2.6 Experimental Results

As in Sects. 8.1.2 and 8.1.3, for our experiments we use a simple, generational genetic algorithm without mutation. For all test problems, tournament selection without replacement of size 2 and uniform crossover is used. Due to the different difficulty of the test problems, we use different population sizes N for the different test instances. The used population sizes N are listed in Table 8.8 and are chosen independently for each test problem. They are chosen such that for some representations the GA finds the optimal solution in more than half of the runs ($P_{succ} > 0.5$). In general, higher N increases the performance (higher P_{succ}), whereas lower N reduces GA performance. However, the performance differences between the different representations remain about the same when using different population sizes N. All runs are stopped after the population is fully converged, or a maximum of 200 generations is reached. We perform 250 runs for each representation and problem instance.

	n	N	Properties of optimal solutions		
			$d_{mst,opt}$	$\min(d_{star,opt})$	$c(T_{opt})$
palmer6	6	16	1	2	693,180
palmer12	12	300	5	7	3,428,509
palmer24	24	800	12	17	1,086,656
raidl10	10	70	3	4	53,674
raidl20	20	800	4	14	157,570
berry6	6	16	0	2	534
berry35u	35	2,000	-	28	16,273
berry35	35	300	0	30	16,915
rothlauf1	16	800	7	9	60,883
rothlauf2	15	2,000	4	8	58,619
rothlauf3	16	1,200	6	9	28,451
rothlauf4	16	800	9	7	112,938

Table 8.8. Population size N used for the different test problems and properties of the optimal (or known best) solutions T_{opt} of the test instances

Furthermore, Table 8.8 presents the properties of the optimal (or best found) solutions T_{opt} for the test problems. The results show that the optimal solutions have about the same minimal distances towards a star than randomly created unbiased individuals (compare Tables 8.6 and 8.7). However, the optimal solutions have a smaller distance towards the MSTs than a randomly generated individual. This means that the good solutions are biased towards the MST. As a result, we expect GAs that use encodings which introduce a modest bias towards the MST to have high performance. For these types of encodings (like the LB ($P_1 = 1$) encoding), MST-like individuals are overrepresented.

The performance of GAs using different representations (compare Sect. 8.1.1) for the different test instances is shown in Table 8.9. We show the mean and standard deviation of the distance $d_{bestf,opt}$ between the best found solution and the optimal solution, the percentage P_{succ} of runs that find the

optimal solution T_{opt}, the mean and standard deviation of the cost $c(T_{bestf})$ of the best solution T_{bestf} that have been found after t_{conv} generations, and the mean and standard deviation of the number of generations t_{conv} until the population is fully converged or the GA run is stopped after 200 generations. $d_{bestf,opt}$ and P_{succ} are related as a high percentage P_{succ} results in a low distance $d_{bestf,opt}$. All results are averaged over 250 runs.

Table 8.9. Performance of GA using different types of representations for the test problems described in Sect. 8.2.3.

		$d_{bestf,opt}$	P_{succ}	$c(T_{bestf})$	t_{conv}
		μ (σ)		μ (σ)	μ (σ)
palmer6 ($c(T_{opt}$=693,180)	Prüfer	1.92 (0.93)	0.056	733,820 (39,464)	14.1 (3.5)
	NetKey	1.32 (1.03)	0.268	716,171 (26,364)	15.4 (3.3)
	CV	1.09 (1.02)	0.392	706,908 (17,693)	18.4 (5.6)
	NB ($P_2 = 1$)	0.38 (0.71)	0.720	697,842 (12,041)	10.7 (2.8)
	NB ($P_2 = 20$)	2.66 (0.68)	0.004	780,191 (36,189)	7.7 (3.1)
	LB ($P_1 = 1$)	0.56 (0.85)	0.660	698,782 (8,561)	12.1 (2.6)
	LB ($P_1 = 20$)	1.14 (0.97)	0.328	708,900 (18,926)	15.2 (3.3)
	LNB (P_1=P_2=1)	0.70 (0.88)	0.532	700,173 (12,790)	16.2 (3.5)
	NetDir	1.57 (0.83)	0.088	727,717 (42,784)	14.3 (3.8)
	KrukalRST	1.62 (1.00)	0.152	731,675 (48,865)	17.1 (5.4)
	KruskalRST*	1.58 (0.95)	0.136	728,206 (36,666)	10.6 (2.6)
	heur. ini	1 (0)	0	709,770 (0)	1.5 (0.6)
	heur. Xover	1 (0.70)	0.224	711,885 (21,340)	6.3 (1.8)
	heur. ini & xover	1 (0)	0	709,770 (0)	1.1 (0.3)
palmer12 ($c(T_{opt}$=3,428,509)	Prüfer	6.49 (0.95	0	4,111,091 (100,102)	46.1 (6.9)
	NetKey	2.03 (1.78)	0.360	3,452,988 (27,768)	60.4 (7.2)
	CV	4.10 (1.16)	0	3,700,839 (115,314)	200 (0)
	NB ($P_2 = 1$)	0 (0)	1	3,428,509 (0)	35.5 (5.2)
	NB ($P_2 = 20$)	1.09 (1.52)	0.604	3,463,583 (56,789)	33.6 (6.7)
	LB ($P_1 = 1$)	0.73 (1.18)	0.628	3,434,632 (14,913)	52.6 (7.0)
	LB ($P_1 = 20$)	1.99 (1.76)	0.352	3,450,763 (27,265)	60.2 (7.9)
	LNB (P_1=P_2=1)	1.78 (1.64)	0.432	3,448,563 (22,482)	77.3 (11.1)
	NetDir	1.72 (1.55)	0.268	3,456,236 (38,339)	61.5 (8.8)
	KrukalRST	2.53 (1.32)	0.068	3,462,151 (44,551)	88.5 (10.7)
	KruskalRST*	1.90 (1.52)	0.292	3,449,270 (22,488)	36.5 (8.3)
	heur. ini	6.00 (0)	0	3,727,552 (0)	10.7 (1.0)
	heur. Xover	4.81 (0.39)	0	3,744,861 (36,297)	16.7 (2.9)
	heur. ini & xover	5.00 (0)	0	3,876,488 (0)	4.4 (0.6)

		$d_{bestf,opt}$	P_{succ}	$c(T_{bestf})$	t_{conv}
		μ (σ)		(σ)	μ (σ)
palmer24 ($c(T_{opt}$=1,086,656)	Prüfer	7.46 (1.06)	0	1,584,450 (171,348)	90.7 (22.9)
	NetKey	0.26 (0.44)	0.736	1,086,866 (353)	87.3 (7.8)
	CV	12.48 (1.59)	0	2,769,489 (286,982)	200 (0)
	NB ($P_2 = 1$)	2.52 (0.73)	0	1,120,965 (40,601)	55.3 (11.2)
	NB ($P_2 = 20$)	10.07 (2.74)	0	1,627,326 (196,475)	40.2 (10.9)
	LB ($P_1 = 1$)	0.56 (0.56)	0.476	1,087,090 (436)	85.4 (8.8)
	LB ($P_1 = 20$)	0.26 (0.44)	0.740	1,086,863 (350)	86.6 (9.2)
	LNB (P_1=P_2=1)	0.16 (0.39)	0.852	1,092,012 (16,559)	135.3 (16.2)
	NetDir	4.10 (1.84)	0	1,119,077 (43,933)	199.8 (1.3)
	KrukalRST	10.65 (1.34)	0	1,547,827 (91,086)	200 (0)
	KruskalRST*	1.80 (0.89)	0.068	1,088,152 (778)	54.3 (2.3)
	heur. ini	11.00 (0)	0	1,884,444 (0)	6.9 (0.4)
	heur. Xover	10.99 (0.32)	0	1,855,160 (76,119)	21.7 (3.6)
	heur. ini & xover	12.00 (0)	0	1,959,790 (0)	6.7 (0.7)
raidl10 ($c(T_{opt}$=53,674)	Prüfer	1.81 (0.89)	0.080	68,867 (9,914)	29.0 (5.2)
	NetKey	0.34 (0.71)	0.760	54,514 (1,967)	33.1 (4.6)
	CV	0.16 (0.38)	0.844	53,876 (520)	81.2 (16.7)
	NB ($P_2 = 1$)	0 (0.06)	0.996	53,693 (308)	16.6 (3.3)
	NB ($P_2 = 20$)	1.03 (0.49)	0.052	59,262 (5,361)	16.8 (4.1)
	LB ($P_1 = 1$)	0 (0)	1	53,674 (0)	24.8 (3.7)
	LB ($P_1 = 20$)	0.27 (0.59)	0.788	54,363 (1,776)	32.0 (4.4)
	LNB (P_1=P_2=1)	0.04 (0.21)	0.956	53,888 (999)	30.9 (4.9)
	NetDir	0.26 (0.52)	0.772	54,415 (2,275)	32.4 (4.0)
	KrukalRST	0.34 (0.57)	0.708	54,208 (1,398)	51.8 (9.2)
	KruskalRST*	0.79 (0.92)	0.480	56,013 (4,286)	23.1 (3.0)
	heur. ini	2.04 (0.21)	0	55,942 (355)	7.4 (1.4)
	heur. Xover	2.76 (0.45)	0	57,261 (864)	13.6 (3.7)
	heur. ini & xover	3.00 (0)	0	58,352 (0)	2.1 (0.8)
berry6 ($c(T_{opt}$=534)	Prüfer	1.87 (0.69)	0.012	677 (75)	13.4 (3.3)
	NetKey	0.52 (0.67)	0.568	557 (36)	14.5 (3.1)
	CV	0.35 (0.62)	0.720	549 (33)	16.9 (4.0)
	NB ($P_2 = 1$)	0.04 (0.22)	0.960	535 (7)	9.0 (2.7)
	NB ($P_2 = 20$)	1.78 (0.67)	0.016	628 (59)	9.8 (2.9)
	LB ($P_1 = 1$)	0.02 (0.13)	0.984	535 (5)	9.2 (2.2)
	LB ($P_1 = 20$)	0.46 (0.61)	0.604	553 (29)	14.0 (3.0)
	LNB (P_1=P_2=1)	0.09 (0.29)	0.908	536 (8)	13.2 (3.3)
	NetDir	0.90 (0.86)	0.380	588 (69)	13.9 (3.1)
	KrukalRST	0.63 (0.75)	0.524	563 (41)	15.2 (3.7)
	KruskalRST*	1.08 (0.94)	0.324	592 (68)	10.3 (2.2)
	heur. ini	0 (0)	1	534 (0)	0.7 (0.6)
	heur. Xover	0.22 (0.45)	0.800	545 (26)	5.4 (1.1)
	heur. ini & xover	0 (0)	1	534 (0)	0.6 (0.5)

		$d_{best,opt}$	P_{succ}	$c(T_{best})$	t_{conv}
		μ (σ)		(σ)	μ (σ)
berry35 ($c(T_{opt}$=16,915)	Prüfer	25.37 (1.25)	0	59,553 (2,795)	104.7 (12.3)
	NetKey	4.11 (1.66)	0.028	18,083 (763)	99.1 (8.6)
	CV	24.51 (1.69)	0	120,397 (12,749)	200 (0)
	NB ($P_2 = 1$)	0.16 (0.36)	0.988	16,916 (12)	56.1 (6.4)
	NB ($P_2 = 20$)	8.58 (2.83)	0	20,769 (1,976)	45.6 (5.9)
	LB ($P_1 = 1$)	0.44 (0.50)	1	16,915 (0)	75.2 (6.8)
	LB ($P_1 = 20$)	2.78 (1.34)	0.112	17,519 (526)	94.0 (8.0)
	LNB (P_1=P_2=1)	1.88 (1.09)	0.256	17,181 (250)	128.2 (12.7)
	NetDir	14.06 (2.28)	0	27,975 (4,019)	200 (0)
	KrukalRST	18.47 (1.93)	0	39,806 (3,512)	200 (0)
	KruskalRST*	6.87 (1.96)	0	19,913 (1,268)	74.8 (4.5)
	heur. ini	0 (0)	1	16,915 (0)	0 (0)
	heur. Xover	0.03 (0.18)	0.972	16,923 (56)	15.0 (0.7)
	heur. ini & xover	0 (0)	1	16,915 (0)	0 (0)
berry35u ($c(T_{opt}$=16,273)	Prüfer	30.68 (0.62)	0	21,553 (230)	190.2 (29.1)
	NetKey	8.88 (2.01)	0	17,111 (326)	166.6 (30.2)
	CV	25.94 (1.61)	0	34,989 (1,558)	200 (0)
	NB ($P_2 = 1$)	30 (0)	0	21,513 (0)	27.3 (3.7)
	NB ($P_2 = 20$)	30 (0)	0	21,513 (0)	28.5 (4.1)
	LB ($P_1 = 1$)	8.48 (2)	0	17,061 (329)	169.3 (28.9)
	LB ($P_1 = 20$)	8.54 (1.99)	0	17059 (362)	169 (30.5)
	LNB (P_1=P_2=1)	25.56 (1.62)	0	20,093 (353)	189.4 (18.1)
	NetDir	20.56 (1.98)	0	22,264 (1,441)	200 (0)
	KrukalRST	22.82 (2.34)	0	26,929 (1053)	200 (0)
	KruskalRST*	7.08 (0.97)	0.42	16,299 (73.2)	131.22 (47.9)
	heur. ini	32.56 (1.03)	0	42,129 (5,711)	79.48 (87.48)
	heur. Xover	7.1 (0.86)	0.32	16,323 (74.7)	125.58 (44.84)
	heur. ini & xover	32.74 (1.07)	0	42,803 (6,585)	91.4 (89.87)
rothlauf1 ($c(T_{opt}$=60,883)	Prüfer	6.94 (0.42)	0	63,680 (236)	62.1 (14.1)
	NetKey	1.34 (1.16)	0.296	61,144 (345)	86.6 (9.1)
	CV	8.75 (1.20)	0	71,657 (2,592)	200 (0)
	NB ($P_2 = 1$)	8.47 (0.97)	0	64,654 (684)	34.2 (9.6)
	NB ($P_2 = 20$)	8.98 (0.20)	0	64,997 (106)	28.6 (5.8)
	LB ($P_1 = 1$)	1.39 (1.22)	0.312	61,202 (424)	85.1 (8.5)
	LB ($P_1 = 20$)	1.50 (1.21)	0.272	61,161 (372)	85.5 (8.4)
	LNB (P_1=P_2=1)	2.77 (0.95)	0.004	61,233 (123)	119.3 (17.5)
	NetDir	3.88 (1.24)	0	61,709 (523)	110.9 (11.1)
	KrukalRST	7.91 (1.47)	0	64,539 (1,411)	200 (0)
	KruskalRST*	1.19 (1.08)	0.332	61,315 (474)	54.2 (5.0)
	heur. ini	9.00 (0)	0	61,817,796 (0)	6.0 (0.3)
	heur. Xover	7.25 (0.62)	0	628,632 (182,178)	20.9 (3.1)
	heur. ini & xover	7.00 (0)	0	71,864,456 (0)	5.1 (0.5)

		$d_{bestf,opt}$	P_{succ}	$c(T_{bestf})$	t_{conv}
		μ (σ)		(σ)	μ (σ)
rothlauf2 ($c(T_{opt}$=58,619)	Prüfer	8.16 (0.91)	0	66,221 (868)	63.9 (28.2)
	NetKey	0.46 (0.83)	0.756	58,651 (57)	91.1 (9.0)
	CV	7.55 (1.23)	0	69,598 (1,960)	200 (0)
	NB ($P_2 = 1$)	8.74 (1.52)	0	69,901 (2,421)	45.7 (14.2)
	NB ($P_2 = 20$)	9.93 (0.49)	0	73,276 (917)	34.4 (8.1)
	LB ($P_1 = 1$)	0.54 (0.88)	0.720	58,664 (114)	90.9 (9.6)
	LB ($P_1 = 20$)	0.43 0.79)	0.760	58,652 (61)	90.2 (8.5)
	LNB (P_1=P_2=1)	2.06 (0.33)	0	59,078 (318)	132.3 (18.1)
	NetDir	1.43 (0.98)	0.168	58,872 (231)	107.9 (7.2)
	KrukalRST	2.50 (1.01)	0.004	59,224 (404)	200 (0)
	KruskalRST*	0.17 (0.48)	0.868	58,680 (207)	50.3 (10.1)
	heur. ini	6.00 (0)	0	1,281,721 (0)	6.2 (0.4)
	heur. Xover	4.82 (0.40)	0	74,522 (931)	20.0 (2.9)
	heur. ini & xover	5.00 (0)	0	1,284,189 (0)	8.7 (0.8)
rothlauf3 ($c(T_{opt}$=28,451)	Prüfer	6.94 (0.84)	0	31,247 (500)	79.0 (23.0)
	NetKey	1.31 (1.49)	0.440	28,722 (352)	98.2 (9.1)
	CV	9.32 (1.37)	0	34,870 (936)	200 (0)
	NB ($P_2 = 1$)	8.17 (1.16)	0	32,704 (897)	45.7 (16.2)
	NB ($P_2 = 20$)	8.92 (0.38)	0	34,286 (573)	34.3 (8.1)
	LB ($P_1 = 1$)	1.42 (1.52)	0.416	28,709 (337)	97.6 (9.8)
	LB ($P_1 = 20$)	1.35 (1.59)	0.480	28,694 (351)	99.4 (10.7)
	LNB (P_1=P_2=1)	4.64 (1.09)	0	30,386 (298)	157.2 (21.3)
	NetDir	1.80 (1.46)	0.384	28,891 (395)	115.2 (13.3)
	KruskalRST	6.24 (1.34)	0	30,190 (520)	200 (0)
	KruskalRST*	2.50 (2.01)	0.240	29,182 (542)	60.3 (10.4)
	heur. ini	8.00 (0)	0	61,781,980 (0)	6.1 (0.3)
	heur. Xover	4.92 (0.28)	0	589,340 (168,997)	19.0 (2.6)
	heur. ini & xover	6.00 (0)	0	71,829,016 (0)	5.0 (0.4)
rothlauf4 ($c(T_{opt}$=112,938)	Prüfer	9.20 (0.7)	0	128,572 (2,350)	68.2 (29.0)
	NetKey	3.20 (1.6)	0.128	115,016 (2968)	94 (8.2)
	CV	10.03 (1.4)	0	146,387 (4,592)	200 (0)
	NB ($P_2 = 1$)	10.30 (0.8)	0	125,095 (1,856)	52.4 (45.7)
	NB ($P_2 = 20$)	11.76 (0.7)	0	128,211 (1,260)	28.6 (17.2)
	LB ($P_1 = 1$)	3.10 (1.5)	0.136	114,741 (2,570)	94 (11.6)
	LB ($P_1 = 20$)	3.22 (1.6)	0.14	115,106 (2,790)	93.6 (11.0)
	LNB (P_1=P_2=1)	5.80 (1.32)	0	117,755 (1,990)	132.1 (21.9)
	NetDir	1.84 (0.9)	0.156	113,086 (438)	92.2 (12.3)
	KrukalRST	5.57 (1.0)	0.004	119,860 (941)	200 (0)
	KruskalRST*	4.20 (1.8)	0.056	117,515 (4,107)	63.98 (28.5)
	heur. ini	9 (0)	0	186,230 (0)	8.14 (0.4)
	heur. Xover	8.07 (0.2)	0	156,209 (5,134)	19.61 (2.7)
	heur. ini & xover	9 (0)	0	197,511 (0)	7.32 (0.8)

The results confirm the predictions from Sect. 8.2.5. Due to problems with low locality, GAs using Prüfer numbers fail and do not find the optimal solution even for small problem instances. GAs using the uniformly and synonymously redundant NetKey encoding show a good performance for the different test problems independently of the properties of the optimal solution. The CV encoding results in low GA performance as the non-synonymous redundancy of the encoding does not allow guided search and GAs behave like random search. Only for small six or ten node problem instances can good solutions be found, but large numbers of generations are necessary. The results for the LNB encoding and variants of it show that the use of low values for P_1 an P_2 results in high GA performance as solutions similar to the MST are overrepresented. The LB encoding with $P_1 = 1$ seems to be an especially good choice, whereas using the NB encoding with large P_2 results in a strong bias towards stars and does not allow the GA to reliably solve the test problems. The direct encoding NetDir shows good results and allows GAs to reliably solve the problem instances. The situation is different when using edge-sets as the performance of GAs depends on the properties of the optimal solution T_{opt}. GAs using heuristic variants of the edge-set encoding (heur. ini, heur. xover, or heur. ini & xover) can only solve test problems where the optimal solution is the MST ($d_{mst,opt} = 0$). For other test problems, the optimal solution T_{opt} can not be found. In contrast, GAs using the non-heuristic and unbiased variants of the edge-set encoding (KruskalRST or KruskalRST*) often show good performance independently of the structure of T_{opt}.

When focusing on the different test problems, the results reveal that for berry6 and berry35, where the optimal solution is the MST, GAs using either the LNB encoding with low bias values, or edge-sets with heuristics, show very high performance and outperform all other unbiased encodings. However, with increasing $d_{opt,MST}$, GAs using heuristic variants of the edge-set fail as the bias towards the MST is too strong. GAs using the LB or LNB encoding with low link-specific bias still allow efficient search as the overrepresentation of MST-like solutions is lower than the bias of the heuristic variants of the edge-sets. Furthermore, the results show that the performance of GAs using uniformly redundant or unbiased encodings such as NetKeys, NetDir, edge-sets without heuristics, or the LB ($P_1 = 20$) encoding is nearly independent of the structure of the optimal solution T_{opt}. P_{succ} and t_{conv} are not strongly affected by the properties of the optimal solution.

We see that the empirical results confirm the theoretical predictions made in Sect. 8.2.5. As expected, it is more difficult to predict the performance of direct representations as properties of problem-specific search operators (like a possible bias) are more difficult to identify. However, using the developed theory of representations for a thorough analysis of search operators (compare Sect. 7.2 regarding the edge-sets) allows us to validate the predictions about the expected GA performance not only for indirect, but also for direct representations.

The experimental results confirmed the results of the theoretical analysis from the previous chapters and showed that only NetKeys, the LNB encoding with a large link-specific bias P_1, edge-sets without heuristics, and NetDir show good performance for the test problems independently of the properties of the optimal solution. Furthermore, as expected, GAs using CVs fail for large problem instances due to problems with non-synonymous redundancy, GAs using Prüfer number fail due to the low locality of the encoding, GAs using the NB encoding show low performance due to their bias towards stars, and GAs using heuristic variants of the edge-set encoding fail if the optimal solution is not the MST.

8.3 Summary

This chapter investigated for different tree problems how representations influence the performance of GEAs. Section 8.1 focused on scalable test problems like the one-max tree problem and the deceptive trap problem for trees, and examined how the performance of GEAs depends on the used representation. These two types of test problems for trees allow us to determine a priori the structure of the optimal solution. The experimental results confirmed the theoretical predictions regarding the influence of representations on the performance of GAs. Section 8.2 focused on OCST problems and investigated for different test problems from the literature how the used representation influences GEA performance. We defined the problem, described existing approaches to solve the problem, and developed theoretical predictions on how the different representations influence GEA performance based on the properties of the encodings from Chaps. 6 and 7. The experimental results confirmed the theoretical predictions.

This chapter illustrated how GEA users can use the framework of representations for predicting and explaining the performance of GEAs using different types of representations. When applying GEAs to the one-max tree and deceptive trap problem, the optimal solutions are known a priori and it is possible to validate the predictions concerning the expected GEA performance. The experimental results for the scalable test problems confirmed the theoretical predictions. GEAs using Prüfer numbers fail for one-max tree problems but perform well for deceptive tree problems. Due to problems with non-synonymous redundancy, GEAs using CVs can only solve small instances of the one-max tree problem. When using CVs for larger one-max tree problems, GEAs fail. The situation is different for deceptive trap problems as the non-synonymity of CVs makes such problems easier and allows GAs to find optimal solutions more easily in comparison to other encodings. LNB encodings with a small link-specific or node-specific bias are biased either towards stars or towards the MST. Therefore, GEAs using such encodings perform well if the optimal solution is either star-like or MST-like. If the optimal solution is not star-like or MST-like, GEAs using LNB encodings with a small bias

show low performance. The situation is similar for edge-sets. Edge-sets with heuristics show a strong bias towards the MST and only perform well if the optimal solution is the MST. If the optimal solution is slightly different from the MST, GEAs using edge-sets with heuristics fail. Edge-sets without heuristics are nearly unbiased and GA performance is independent of the optimal solution. Network random keys show the same performance as LNB encodings with a large link-specific bias P_1. GEAs using these encodings perform reliably well on all test instances and are in general a good choice for the one-max tree and deceptive tree problem.

The situation is similar when using GEAs for OCST problems. For many problem instances, the optimal solution is similar to the MST. Therefore, GEAs using encodings that are biased towards the MST show good performance. However, the results show that the bias of the heuristic variants of the edge-sets is too strong, as only problems can be solved where the optimal solution is the MST. A better choice are the LNB or LB encoding with a small link-specific bias ($P_1 = 1$) as such encodings overrepresent solutions similar to the MST but still allow us to find solutions that are different from the MST. Furthermore, the results show that NetKeys and LNB encoding with a large link-specific bias perform well, whereas Prüfer numbers, CVs and LNB encodings with a large node-specific bias fail.

This chapter nicely illustrated the benefits we gain from analyzing the properties of encodings and using the framework about representations for predicting the expected GEA performance. We want to encourage researchers to use the proposed tools for analyzing the properties of representations or search operators for other problem domains. Examining redundancy, scaling, and locality of an encoding a priori allows us to predict GEA performance based on the presented framework about representations.

9

Summary and Conclusions

The purpose of this book is to understand the influence of representations on the performance of genetic and evolutionary algorithms. This chapter summarizes the work contained in this study and lists its major contributions.

9.1 Summary

We started in Chap. 2 by providing the necessary background for examining representations for GEAs. Researchers recognized early that representations have a large influence on the performance of GEAs. Consequently, after a brief introduction into representations and GEAs, we discussed how the influence of representations on problem difficulty can be measured. The chapter ended with prior guidelines for choosing high-quality representations. Most of them are mainly based on empirical observations and intuition and not on theoretical analysis.

Therefore, we presented in Chap. 3 three aspects of a theory of representations for GEAs. We investigated how the locality, scaling, and locality of an encoding influences GEA performance. The performance of GEAs is determined by the solution quality at the end of a run and the number of generations until the population is converged. Consequently, for redundant and exponentially scaled encodings, we presented population sizing models and described how the time to convergence is changed. Furthermore, we were able to demonstrate that high-locality encodings do not change the difficulty of a problem; in contrast, when using low-locality encodings, on average, the difficulty of problems changes. Therefore, easy problems become more difficult and difficult problems become easier by the use of low-locality encodings. For all three properties of encodings, the theoretical models were verified with empirical results.

In Chap. 4, we combined the three elements of representation theory from Chap. 3 to form a framework for a theory-guided design and analysis of representations. The framework describes how the time to convergence and the

solution quality of GEAs depends on the redundancy, scaling, and locality of a representation. Using this framework, we could estimate the performance of GEAs using different types of representations in a theory-guided manner. We recognized that different types of representations can dramatically change the behavior of GEAs. Consequently, we presented major implications when using different types of representations.

In Chaps. 5, 6, and 7, we used the framework for the analysis of existing representations and the design of new representations in a theory-guided matter. We analyzed the redundancy, scaling, and locality of a variety of different representations and made predictions regarding GEA performance based on the framework. These predictions were verified by experimental results. In Chap. 5, we focused on binary representations for integers. We compared the performance of binary, Gray, and unary encodings for integer problems. Based on the properties of the encodings, the framework was able to explain existing performance differences.

Chapter 6 focused on the analysis and design of tree representations. In analogy to Chap. 5, we analyzed the properties of tree representations, namely Prüfer numbers, NetKeys, characteristic vector encodings, and the link-and-node-biased encoding. Based on these investigations, we were able to predict GEA performance using the representation framework. Based on the insights into the insufficiencies of existing tree representations, we combined in Sect. 6.5 advantageous elements of the CV and the LNB encoding to form the NetKey encoding. The NetKey encoding is a redundant encoding that allows GEAs to perform well independently of the structure of the optimal solution. Furthermore, for the one-max tree problem, we presented a population sizing model for the NetKey encoding and verified that the time to convergence depends linearly on the problem size.

Direct representations do not use an explicit genotype-phenotype mapping, but define search operators directly on the phenotypes. Therefore, in general, standard genetic operators can no longer be used and problem-specific operators for the phenotypes must be developed. In Chap. 7, we focused on the analysis and design of direct encodings. We developed search operators for a new direct encoding (NetDir) and performed an exhaustive investigation into the properties of the edge-set encoding. The investigation performed into the properties of the edge-set encoding illustrated that the framework for the design of representations can not only be used for indirect representations, but also for search operators and direct encodings.

Finally, Chap. 8 illustrated how GEA users can use the provided representation framework for estimating the influence of different representations on GEA performance. Based on the analysis of redundancy, scaling, and locality of tree representations, we compared GEAs performance for the one-max tree problem, the deceptive trap problem for trees, and various instances of the optimal communication spanning tree problem.

9.2 Conclusions

We summarize the most important contributions of this work.

Framework for design and analysis of representations (and operators) for GEAs. The main purpose of this study was to present a framework which describes how genetic representations influence the performance of GEAs. The performance of GEAs is measured by the solution quality at the end of the run and the number of generations until the population is converged. The proposed framework allows us to analyze the influence of existing representations on GEA performance and to develop efficient new representations in a theory-guided way. Furthermore, we illustrated that the framework can also be used for the design and analysis of search operators, which are relevant for direct encodings. Based on the framework, the development of high-quality representations remains not only a matter of intuition and random search but becomes an engineering design task. Even though more work is needed, we believe that the results presented are sufficiently compelling to recommend increased use of the framework.

Redundancy, Scaling, and Locality. These are the three elements of the proposed framework of representations. We demonstrated that these three properties of representations influence GEA performance and presented theoretical models to predict how solution quality and time to convergence changes. By examining the redundancy, scaling, and locality of an encoding, we are able to predict the influence of representations on GEA performance.

The theoretical analysis shows that the redundancy of an encoding influences the supply of building blocks (BB) in the initial population. r denotes the number of genotypic BBs that represent the best phenotypic BB, and k_r denotes the order of redundancy. For synonymously redundant encodings, where all genotypes that represent the same phenotype are similar to each other, the probability of GEA failure goes either with $O(\exp(-r/2^{k_r}))$ (uniformly scaled representations) or with $O(\exp(-\sqrt{r/2^{k_r}}))$ (exponentially scaled representations). Therefore, GEA performance increases if the representation overrepresents high-quality BBs. If a representation is uniformly redundant, that means each phenotype is represented by the same number of genotypes, GEA performance remains unchanged in comparison to non-redundant encodings.

The analysis of the scaling of an encoding reveals that non-uniformly scaled representations modify the dynamics of genetic search. If exponentially scaled representations are used, the alleles are solved serially which increases the overall time until convergence and results in problems with genetic drift but allows rough approximations of the expected optimal solution after a few generations.

We know from previous work that the high locality of an encoding is a necessary condition for efficient mutation-based search. An encoding has high locality if neighboring phenotypes correspond to neighboring genotypes. Investigating the influence of locality shows that high-locality encodings do

not change the difficulty of a problem. In contrast, low-locality encodings, where phenotypic neighbors do not correspond to genotypic neighbors, change problem difficulty and make, on average, easy problems more difficult and deceptive problems easier. Therefore, to assure that an easy problem remains easy, high-locality representations are necessary.

Population sizing and time to convergence models for redundant encodings, exponentially scaled encodings, and NetKeys. Based on a better understanding of redundancy and scaling, we were able to formulate population sizing models and time to convergence models for redundant and exponentially scaled encodings. The models show that for redundant encodings the population size grows with $O(2^{k_r}/r)$ and the time to convergence goes with $O(\text{const} - r/2^{k_r k-1})$, where k denotes the order of building blocks. When using exponentially scaled encodings, we have to distinguish whether we want to consider genetic drift or not. When neglecting genetic drift, the population size N is independent of the length l_s of an exponentially scaled BB, but depends only on the number m of competing exponentially scaled BBs ($N = O(\sqrt{m})$). The time to convergence goes with $O(l_s\sqrt{m})$. To consider genetic drift, we developed two different population sizing models (stair-case drift model and approximated drift model) based on the model for the non-drift case. Due to genetic drift, the ability of GEAs to decide well between competing schemata decreases with increasing number of generations. Therefore, the probability of GEA failure increases with larger l_s.

Instead of using binary genotypes, the NetKey encoding uses continuous variables for representing trees. For NetKeys, we presented a population sizing model for the easy one-max tree problem and showed that the population size N goes with $O(n^{1.5})$, where n denotes the number of nodes. The time to convergence is linearly increasing with $O(n)$.

Analysis of binary representations for integers. The framework can be used for explaining the performance differences of GEAs using different types of binary representations for integer problems. The analysis of binary, Gray, and unary encoding has shown that the unary encoding is non-uniformly redundant, that the binary encoding is exponentially scaled, and the Gray and binary encoding have low locality. Therefore, the performance of GEAs using unary encoding depends on the structure of the optimal solution. If the optimal solution is underrepresented, GEAs fail; in contrast, if the optimal solution is overrepresented, GEAs using the unary encoding perform well. When using the binary encoding, all alleles are solved serially and the time to convergence increases. Therefore, some low salient genes are randomly fixed due to genetic drift before they can be reached by the search process. Finally, binary and Gray encoding are low-locality encodings and change the difficulty of the optimization problem. Thus, the resulting problem difficulty depends not only on the used representation but also on the considered optimization problem. Some problems like the easy integer one-max problem are easier when using the binary encoding, but there are other problems that are easier when using the Gray encoding.

Analysis of tree representations. The framework about representations can also be used for analyzing the influence of tree representations and operators on GEA performance. Based on the properties of Prüfer numbers, characteristic vectors, the link and node biased encoding, and edge-sets the proposed framework allowed us to predict GEA performance.

The analysis of the Prüfer number encoding revealed that the locality of the encoding is high around stars and low elsewhere. Therefore, GEA performance is low for easy problems if the optimal solution is not a star, and high if the problem at hand is deceptive and the optimal solution is a non-star.

The link and node biased (LNB) encoding uses a link-specific and node-specific bias to control the representations influence on the structure of the represented phenotype. The investigation into the properties of the encoding revealed that the encoding is synonymously redundant. Furthermore, it is uniformly redundant if a large link-specific bias is used. If the link-specific bias is small the encoding is non-uniformly redundant and biased towards the minimum spanning tree (MST). The use of a node-specific bias results in an additional bias of the encoding towards star-like tree structures. Therefore, only GEAs using LNB encodings with a large link-specific bias perform independently of the structure of the optimal solution. If the link-specific bias is small, GEAs only perform well when searching for optimal MST-like phenotypes. When using an additional node-specific bias, GEAs also perform well when searching for optimal solutions similar to stars.

Analyzing the characteristic vector encoding revealed that the encoding is redundant. Because invalid solutions are possible when using characteristic vectors, an additional repair mechanism is necessary which makes the representation non-synonymously redundant. Due to the uniform redundancy of characteristic vectors, GEA performance is independent of the structure of the optimal solution. However, the repair mechanism results in non-synonymous redundancy which has the same effect as low locality. Therefore, GEA performance is reduced for easy problems and increased for deceptive problems. With increasing problem size, the repair process generates more and more links randomly and offspring trees do not have much in common with their parents. Therefore, for larger problems, guided search is no longer possible and GEAs behave like random search.

The edge-set encoding is an encoding which encodes trees directly by listing their edges. Search operators for edge-sets may be heuristic when considering the weights of edges they include in the offspring, or naive, including edges without regard to their weights. Analyzing the properties of the heuristic variants of the search operators reveals a strong bias towards the MST. Therefore, problems where the optimal solutions are different from the MST could scarcely be solved. In contrast, the naive variants are unbiased which means genetic search is independent of the structure of the optimal solution. Although no explicit genotype-phenotype mapping exists for edge-sets and the framework for the design of representations can not be directly applied, it is useful for structuring the analysis of edge-sets. Similarly to non-uniformly

redundant representations, edge-sets overrepresent some specific types of trees and GEA performance increases if optimal solutions are similar to the MST.

Design of new representations for trees. The framework about representations can not only be used for the analysis of existing representations but also for the development of new representations. Based on the insights into representation theory, the NetKey encoding has been proposed. The NetKey encoding is based on the characteristic vector encoding but uses continuous variables for encoding a tree. The encoding is synonymously and uniformly redundant as well as uniformly scaled. In contrast to other encodings, GEAs using NetKeys are able to distinguish between important and unimportant links. The high performance of GEAs using NetKeys was verified by empirical results and proposes an increasing use of this encoding. Furthermore, we presented the NetDir representation as an example of a theory-guided design of a direct encoding. The search operators for the NetDir representation are developed based on the notion of schemata.

Scalable test problems for trees. Last but not least, we presented the fully easy one-max tree problem and the deceptive trap problem for trees as examples for scalable test problems. For these types of problems the optimal solution is determined a priori and the distance of an individual towards the optimal solution determines the fitness of the individual. Both problems, the one-max tree problem and the deceptive trap problem for trees, are easy to implement and can be advantageously used for comparing the performance of GEAs on tree problems. By providing scalable test instances, the performance of different types of representations or new, more effective search methods can be analyzed and compared more easily.

A

Optimal Communication Spanning Tree Test Instances

Searching the literature for standard test problems for the optimal communication spanning tree (OCST) problem reveals that many researchers use their private test problems which are mostly not published. As a result, the comparison of different search algorithms or representations is a difficult and time consuming task. It is not possible to quickly check if a new search method is better than the existing ones. Furthermore, applicants hesitate to use new and efficient search methods or representations if they can not be tested on a variety of different test problems and solve these problems well and reliably. Therefore, the building up of a collection of test instances for the OCST problem is necessary.

The purpose of this appendix is to go one step in this direction and to present a collection of different test instances for the OCST problem. It gives exact details concerning the properties of the problems we used in Sect. 8.2 for the comparison of different types of representations. Based on the presented test instances, a fair and standardized comparison of new search techniques or representations becomes possible.

For each test problem we present the best known solution, the demands, and the distance weights. The upper right corner of the presented matrices specifies the demands and distance weights. Sect. A.1 summarizes the properties of the test instances from Palmer (1994). We are not able to present data for the 47 and 98 node problems because these are no longer available. Sect. A.2 presents the details for the 10, 20, and 50 node OCST problem from Raidl (2001), Sect. A.3 specifies the berry6, berry35, and berry35u problems presented by Berry et al. (1995), and Sect. A.4 summarizes the specifications of four real-world test problems from Rothlauf et al. (2002).

A.1 Palmer's Test Instances

We present the details for palmer6, palmer12, and palmer24 test instances presented by Palmer (1994).

Table A.1 gives an overview of the best known solutions for the three test problems. The cost of a link is calculated as the traffic over a link multiplied by the distance weight of this link (compare (8.4)). Table A.2 lists the demands and distance weights for the palmer6 problem. The properties of the palmer12 problem are specified in Table A.3 (demands) and Table A.4 (distance weights). Finally, the properties of the palmer24 problem are listed in Table A.5 (demands) and Table A.6 (distance weights).

	$c(T_{opt})$	used links
palmer6	693,180	DET-CHI, PHIL-DET, PHIL-NY, HOU-LA, HOU-DET
palmer12	3,428,509	SD-PHNX, SD-LA, SF-SD, DET-CHI, PHIL-BALT, PHIL-DET, PHIL-NY, DAL-SD, DAL-DET, HOU-DAL, SANAN-DAL
palmer24	1,086,656	LA-PHNX, SD-LA, SF-LA, SJ-LA, CHI-LA, CHI-DEN, DET-CHI, DET-IND, NY-BOS, CLEVE-DET, COL-DET, PHIL-WDC, PHIL-JACK, PHIL-BALT, PHIL-DET, PHIL-NY, HOU-NO, HOU-DET, HOU-MEMP, HOU-DAL, HOU-ELPAS, SANAN-HOU, MILW-CHI

Table A.1. Cost and structure of the best solutions to the test instances from Palmer (1994)

(a) Demand matrix

	LA	CHI	DET	NY	PHIL	HOU
LA	0	1	1	1	1	2
CHI	-	0	10	3	4	3
DET	-	-	0	5	6	2
NY	-	-	-	0	31	2
PHIL	-	-	-	-	0	2
HOU	-	-	-	-	-	0

(b) Distance matrix

	LA	CHI	DET	NY	PHIL	HOU
LA	0	16,661	18,083	21,561	21,099	13,461
CHI	-	0	5,658	9,194	8,797	10,440
DET	-	-	0	7,230	6,899	11,340
NY	-	-	-	0	4,300	13,730
PHIL	-	-	-	-	0	13,130
HOU	-	-	-	-	-	0

Table A.2. Demand and distance matrix for palmer6

Table A.3. Demand matrix for palmer12

	PHNX	LA	SD	SF	CHI	BALT	DET	NY	PHIL	DAL	HOU	SANAN
PHNX	0	7	8	4	2	1	2	1	1	3	3	3
LA	-	0	25	7	1	1	1	1	1	2	2	2
SD	-	-	0	6	1	1	1	1	1	2	2	2
SF	-	-	-	0	1	1	1	1	1	2	2	2
CHI	-	-	-	-	0	4	11	4	4	3	3	2
BALT	-	-	-	-	-	0	6	15	29	2	2	2
DET	-	-	-	-	-	-	0	5	6	3	2	2
NY	-	-	-	-	-	-	-	0	33	2	2	2
PHIL	-	-	-	-	-	-	-	-	0	2	2	2
DAL	-	-	-	-	-	-	-	-	-	0	11	10
HOU	-	-	-	-	-	-	-	-	-	-	0	14
SANAN	-	-	-	-	-	-	-	-	-	-	-	0

Table A.4. Distance matrix for palmer12

	PHNX	LA	SD	SF	CHI	BALT	DET	NY	PHIL	DAL	HOU	SANAN
PHNX	0	6,490	5,903	8,484	14,561	18,359	15,976	19,360	18,867	10,090	10,883	9,665
LA	-	0	4,523	6,256	16,661	20,618	18,083	21,561	21,099	12,639	13,461	12,236
SD	-	-	0	6,908	16,414	20,292	17,829	21,263	20,787	12,073	12,802	11,540
SF	-	-	-	0	17,328	21,452	18,714	22,286	21,874	14,234	15,259	14,136
CHI	-	-	-	-	0	8,425	5,658	9,194	8,797	9,603	10,440	11,237
BALT	-	-	-	-	-	0	6,621	5,067	4,439	12,385	12,526	13,722
DET	-	-	-	-	-	-	0	7,230	6,899	10720	11,340	12,297
NY	-	-	-	-	-	-	-	0	4,300	13,531	13,730	14,912
PHIL	-	-	-	-	-	-	-	-	0	12,967	13,130	14,319
DAL	-	-	-	-	-	-	-	-	-	0	4,888	5,076
HOU	-	-	-	-	-	-	-	-	-	-	0	4,478
SANAN	-	-	-	-	-	-	-	-	-	-	-	0

Table A.5. Demand matrix for palmer24

	PHNX	LA	SD	SF	SJ	DEN	WDC	JACK	CHI	IND	NO	BOS	BALT	DET	NY	CLEVE	COL	PHIL	MEMP	DAL	ELPAS	HOU	SANAN	MILW
PHNX	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LA	-	0	100	100	100	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SD	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SF	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SJ	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DEN	-	-	-	-	-	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
WDC	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0
JACK	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0
CHI	-	-	-	-	-	-	-	-	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	100
IND	-	-	-	-	-	-	-	-	-	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0
NO	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	100	0	0
BOS	-	-	-	-	-	-	-	-	-	-	-	0	0	0	100	0	0	0	0	0	0	0	0	0
BALT	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	100	0	0	0	0	0	0
DET	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	100	100	2	0	0	0	2	0	0
NY	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	2	0	0	0	0	0	0
CLEVE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0
COL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0
PHIL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0
MEMP	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	100	0	0
DAL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	100	0	0
ELPAS	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	100	0	0
HOU	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	100	0
SANAN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0
MILW	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0

Table A.6. Distance matrix for palmer24

	PHNX	LA	SD	SF	SJ	DEN	WDC	JACK	CHI	IND	NO	BOS	BALT	DET	NY	CLEVE	COL	PHIL	MEMP	DAL	ELPAS	HOU	SANAN	MILW
PHNX	0	649	590	848	824	808	1,810	1,700	1,456	1,489	1,336	2,059	1,835	1,597	1,936	1,674	1,588	1,886	1,289	1,009	602	1,088	966	1,461
LA	-	0	453	626	599	980	2,034	1,955	1,661	1,717	1,595	2,275	2,068	1,803	2,151	1,890	1,813	2,109	1,537	1,269	852	1,341	1,226	1,664
SD	-	-	0	698	664	967	2,003	1,897	1,644	1,688	1,539	2,241	2,022	1,789	2,123	1,864	1,784	2,077	1,483	1,203	791	1,282	1,150	1,644
SF	-	-	-	0	282	1,056	2,126	2,106	1,738	1,795	1,758	2,339	2,142	1,874	2,226	1,963	1,892	2,184	1,663	1,424	1,051	1,529	1,416	1,715
SJ	-	-	-	-	0	1,046	2,114	2,088	1,720	1,782	1,742	2,323	2,136	1,860	2,228	1,955	1,887	2,176	1,643	1,404	1,031	1,508	1,390	1,719
DEN	-	-	-	-	-	0	1,451	1,456	1,068	1,129	1,154	1,674	1,471	1,203	1,561	1,295	1,223	1,515	1,005	833	741	975	920	1,065
WDC	-	-	-	-	-	-	0	855	829	754	1,064	675	387	657	524	615	605	463	914	1,206	1,582	1,225	1,348	853
JACK	-	-	-	-	-	-	-	0	1,036	929	752	1,144	888	983	1,005	979	870	942	817	1,037	1,424	954	1,092	1,097
CHI	-	-	-	-	-	-	-	-	0	549	995	1,030	845	568	914	651	600	877	736	963	1,264	1,040	1,127	475
IND	-	-	-	-	-	-	-	-	-	0	919	993	771	567	875	618	528	821	667	932	1,280	995	1,089	592
NO	-	-	-	-	-	-	-	-	-	-	0	1,373	1,100	1,047	1,226	1,074	958	1,161	621	675	1,058	577	714	1,056
BOS	-	-	-	-	-	-	-	-	-	-	-	0	655	823	530	813	8,571	585	1,203	1,494	1,851	1,511	1,631	1,038
BALT	-	-	-	-	-	-	-	-	-	-	-	-	0	661	507	633	627	449	940	1,235	1,617	1,256	1,372	862
DET	-	-	-	-	-	-	-	-	-	-	-	-	-	0	720	463	491	689	812	1,070	1,404	1,130	1,227	576
NY	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	702	722	430	1,068	1,351	1,727	1,370	1,492	934
CLEVE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	493	660	858	1,123	1,472	1,170	1,279	663
COL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	671	739	1,019	1,377	1,060	1,162	635
PHIL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	1,000	1,297	1,663	1,310	1,439	896
MEMP	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	652	1,044	680	795	792
DAL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	744	488	506	990
ELPAS	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	809	674	1,280
HOU	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	448	1,084
SANAN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	1,166
MILW	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0

A.2 Raidl's Test Instances

This section presents the details for the raidl10, raidl20, and raidl50 test problem proposed by Raidl (2001). We do not list the 75 and 100 nodes test problems herein because they are too extensive to be published. However, the details of the test instances are available and can be directly obtained from the author[1]. All demands and distance weights between the nodes were generated randomly and are uniformly distributed. The cost of a link is calculated as the amount of traffic over a link multiplied by its distance weight (compare (8.2)). The nodes are labeled with numbers between zero and $n-1$.

Table A.7 presents the properties of the best known solutions for the raidl10 and raidl20 problems. The demands and distance weights of the raidl10 test problem are specified in Table A.8. The demands for the raidl20 and raidl50 test problem are shown in Tables A.9 and A.11. The corresponding distance weights can be found in Tables A.10 and A.12.

Table A.7. Cost and structure of the best solutions for raidl10 and raidl20

	$c(T_{opt})$	used links
raidl10	53,674	1-0, 2-0, 3-0, 4-1, 5-0, 6-0, 7-3, 8-1, 9-1
raidl20	157,570	2-0, 7-5, 9-6, 9-7, 10-0, 11-0, 12-4, 13-0, 13-1, 13-3, 13-4, 14-10, 16-2, 17-0, 17-15, 18-8, 18-9, 18-10, 19-10

Table A.8. Demand and distance matrix for raidl10

(a) Demand matrix

	0	1	2	3	4	5	6	7	8	9
0	0	34	97	50	93	100	89	24	89	3
1	-	0	79	65	78	81	82	66	98	72
2	-	-	0	11	36	87	23	78	97	81
3	-	-	-	0	23	88	40	91	83	84
4	-	-	-	-	0	80	16	47	96	9
5	-	-	-	-	-	0	46	84	100	0
6	-	-	-	-	-	-	0	53	78	66
7	-	-	-	-	-	-	-	0	98	58
8	-	-	-	-	-	-	-	-	0	13
9	-	-	-	-	-	-	-	-	-	0

(b) Distance matrix

	0	1	2	3	4	5	6	7	8	9
0	0	8	17	1	41	12	7	16	90	47
1	-	0	47	31	17	87	59	14	5	9
2	-	-	0	53	36	29	47	14	18	84
3	-	-	-	0	53	83	72	6	79	36
4	-	-	-	-	0	64	39	52	16	31
5	-	-	-	-	-	0	63	75	47	5
6	-	-	-	-	-	-	0	21	45	87
7	-	-	-	-	-	-	-	0	89	31
8	-	-	-	-	-	-	-	-	0	45
9	-	-	-	-	-	-	-	-	-	0

[1]Address: Günther Raidl, Institute of Computer Graphics, Vienna University of Technology, Favoritenstraße 9-11/1861, 1040 Vienna, Austria. E-Mail: raidl@apm.tuwien.ac.at

Table A.9. Demand matrix for raidl20

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	19	7	97	99	22	65	82	53	76	2	44	7	40	67	100	7	40	94	90
1	-	0	37	18	98	75	99	90	42	51	4	91	76	91	10	49	53	75	72	17
2	-	-	0	56	91	59	24	34	33	30	0	32	38	6	25	94	43	9	57	18
3	-	-	-	0	8	13	26	25	17	16	67	74	93	16	26	33	54	10	90	44
4	-	-	-	-	0	69	80	44	1	10	10	100	14	16	92	7	26	0	30	44
5	-	-	-	-	-	0	75	43	36	66	26	18	33	100	11	15	26	44	69	2
6	-	-	-	-	-	-	0	100	79	37	80	22	39	56	32	4	70	48	96	77
7	-	-	-	-	-	-	-	0	74	63	73	84	3	16	86	70	8	4	2	8
8	-	-	-	-	-	-	-	-	0	82	84	0	92	52	2	58	30	39	3	18
9	-	-	-	-	-	-	-	-	-	0	44	59	50	15	28	64	77	71	4	5
10	-	-	-	-	-	-	-	-	-	-	0	43	88	9	25	40	79	34	44	47
11	-	-	-	-	-	-	-	-	-	-	-	0	8	92	30	8	83	82	77	40
12	-	-	-	-	-	-	-	-	-	-	-	-	0	78	82	43	96	93	68	11
13	-	-	-	-	-	-	-	-	-	-	-	-	-	0	7	96	75	84	66	79
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	90	76	33	99	0
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	73	43	0	83
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	90	8	74
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	86	83
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	22
19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0

Table A.10. Distance matrix for raidl20

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	30	6	15	67	100	99	34	85	56	3	13	23	2	72	24	53	12	94	57
1	-	0	98	30	53	35	38	59	85	82	85	78	16	3	59	73	17	77	73	15
2	-	-	0	62	9	70	65	21	44	18	44	68	71	56	13	79	5	43	83	39
3	-	-	-	0	80	93	23	61	78	52	28	80	62	1	48	38	25	62	100	33
4	-	-	-	-	0	84	16	6	27	85	49	46	7	4	59	37	8	53	19	98
5	-	-	-	-	-	0	32	12	20	92	41	71	20	72	32	72	19	22	96	80
6	-	-	-	-	-	-	0	73	80	15	88	85	94	72	34	39	79	89	49	15
7	-	-	-	-	-	-	-	0	49	1	86	46	32	97	66	76	37	88	47	8
8	-	-	-	-	-	-	-	-	0	72	18	78	93	65	57	65	44	24	4	29
9	-	-	-	-	-	-	-	-	-	0	17	75	14	55	5	54	56	72	2	56
10	-	-	-	-	-	-	-	-	-	-	0	57	100	40	5	17	67	93	4	13
11	-	-	-	-	-	-	-	-	-	-	-	0	52	75	82	29	19	46	37	83
12	-	-	-	-	-	-	-	-	-	-	-	-	0	41	60	38	21	76	13	86
13	-	-	-	-	-	-	-	-	-	-	-	-	-	0	83	69	40	90	40	93
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	97	48	92	36	52
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	8	2	44	12
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	66	95	38
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	99	23
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	57
19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0

Table A.11. Demand matrix for raidl50

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
0	0	62	56	24	24	29	17	9	97	47	42	5	25	22	11	59	16	97	87	26	65	7	38	9	66	74	94	2	23	14	65	42	42	87	32	66	16	49	75	79	97	83	50	21	71	62	47	87	24	100
1	-	0	79	55	6	16	64	73	57	23	41	80	37	72	88	45	58	19	78	40	69	18	86	31	1	35	18	38	63	65	25	54	64	70	8	37	87	39	9	43	62	50	22	66	21	10	10	45	96	88
2	-	-	0	86	30	73	37	61	40	72	46	78	1	10	2	55	41	73	29	78	25	68	53	68	97	2	56	62	90	32	38	34	95	93	86	24	65	89	52	4	27	98	82	28	74	51	49	14	90	44
3	-	-	-	0	58	14	12	10	48	75	79	70	36	68	69	40	69	63	32	20	53	97	9	4	67	36	1	49	30	42	66	79	22	55	22	81	35	0	91	83	41	36	18	77	3	53	84	38	15	15
4	-	-	-	-	0	25	69	79	0	73	45	36	41	60	32	49	92	77	71	46	65	51	81	66	8	29	6	44	14	50	13	67	33	18	49	14	43	17	59	43	56	71	45	63	97	77	11	89	53	83
5	-	-	-	-	-	0	34	17	100	82	49	7	77	22	17	91	38	98	24	37	15	73	51	24	56	77	67	78	47	78	41	43	54	18	31	73	67	32	90	67	13	5	40	56	27	58	13	65	55	37
6	-	-	-	-	-	-	0	1	36	9	18	60	31	95	93	8	7	70	15	51	90	34	48	62	0	80	17	33	59	22	40	82	15	98	95	46	18	31	13	20	6	32	80	37	93	72	45	67	7	61
7	-	-	-	-	-	-	-	0	84	97	95	31	58	61	78	41	61	2	64	0	84	45	64	78	92	48	75	4	68	81	2	13	17	62	51	63	28	58	90	11	54	50	8	78	10	86	18	37	55	48
8	-	-	-	-	-	-	-	-	0	37	38	94	0	83	85	14	57	55	82	38	58	95	21	19	45	50	13	69	39	91	22	89	99	66	66	51	85	2	5	32	6	9	92	73	92	76	88	15	31	69
9	-	-	-	-	-	-	-	-	-	0	19	55	30	40	40	41	91	53	77	96	9	65	85	7	31	16	58	82	85	29	80	91	39	72	30	97	13	17	11	10	52	30	65	48	37	71	90	27	90	66
10	-	-	-	-	-	-	-	-	-	-	0	22	99	30	73	6	27	89	30	8	40	60	89	30	99	26	60	95	39	43	73	50	96	2	81	43	39	52	99	32	41	30	21	40	27	94	46	54	49	76
11	-	-	-	-	-	-	-	-	-	-	-	0	29	89	35	84	85	100	9	45	61	48	54	33	64	49	1	45	59	7	63	57	39	3	88	60	43	14	20	55	34	69	98	63	57	99	46	41	65	21
12	-	-	-	-	-	-	-	-	-	-	-	-	0	52	25	36	6	58	100	21	25	10	80	32	73	37	72	43	91	98	52	71	17	7	4	86	71	34	8	69	46	16	33	68	68	24	70	40	48	35
13	-	-	-	-	-	-	-	-	-	-	-	-	-	0	62	74	46	41	5	85	44	43	27	34	7	46	4	24	53	76	77	90	9	85	24	55	0	58	89	35	48	58	75	97	94	36	70	5	44	41
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	90	88	51	84	89	58	29	93	82	48	68	58	3	77	9	27	99	76	51	87	10	100	11	86	96	71	88	31	76	98	72	31	86	22	81
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	74	80	76	32	28	23	67	52	26	9	61	20	7	3	71	61	13	70	38	65	65	8	53	96	84	50	34	14	1	22	62	41	69	37
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	74	63	27	40	14	19	15	42	39	23	45	10	50	24	80	88	90	11	62	42	73	45	58	73	25	60	96	87	0	30	24	40	93
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	51	46	73	70	62	14	76	51	25	86	67	50	65	20	39	42	82	47	15	93	4	88	17	30	49	71	31	79	61	37	71	78
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	84	44	13	11	24	89	28	50	74	95	66	5	14	4	13	62	51	28	54	21	83	71	52	31	7	49	10	68	86	47	11
19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	35	91	25	12	15	13	40	31	54	34	97	25	14	67	38	76	84	33	29	4	15	67	22	46	74	37	22	8	90	70
20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	19	24	60	10	3	41	91	43	38	10	44	0	35	58	67	39	0	50	72	29	21	53	62	9	100	2	47	21	10
21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	36	57	96	26	17	73	29	24	63	73	29	73	83	29	74	40	97	12	40	12	51	36	100	3	98	9	69	100
22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	56	91	76	92	47	38	17	30	10	47	21	39	86	50	11	68	45	85	74	7	63	81	20	13	83	19	84
23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	46	95	52	12	50	8	54	7	22	92	25	52	68	38	39	6	90	55	84	23	0	34	63	74	98	9
24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	94	77	92	80	26	4	40	79	16	91	87	70	98	8	62	89	27	96	93	32	2	82	88	52	71
25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	54	87	34	27	50	43	88	93	1	33	19	5	73	98	21	29	50	57	94	59	85	82	52	81
26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	41	84	49	89	37	0	60	91	87	60	85	36	2	38	29	70	71	14	41	9	11	62	39
27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	61	86	99	86	70	46	37	50	87	88	99	76	24	66	35	15	52	95	66	55	63	3
28	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	50	100	40	64	40	49	41	2	54	68	88	52	54	23	99	57	74	51	44	38	26
29	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	69	70	27	50	89	21	82	43	51	85	59	50	91	89	90	39	29	58	60	97
30	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	11	78	16	35	42	40	75	94	84	79	86	18	49	13	34	37	0	15	46
31	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	51	66	71	67	56	59	23	96	54	81	21	16	59	99	100	60	7	39
32	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	34	0	22	79	86	7	94	65	41	97	66	23	8	83	89	79	16
33	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	45	3	39	6	23	86	27	40	44	92	39	3	99	44	3	65
34	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	66	83	17	39	42	48	81	39	13	3	47	63	92	26	45
35	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	2	96	84	8	86	35	1	25	80	60	30	49	24	74
36	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	19	90	5	68	73	45	9	20	92	48	0	61	96
37	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	63	18	88	74	21	49	57	96	34	92	98	25
38	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	37	57	21	53	81	61	72	36	67	5	8
39	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	78	14	29	69	63	96	96	24	24	13
40	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	78	98	34	26	54	30	27	11	94
41	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	52	15	50	74	34	97	34	72
42	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	100	67	77	7	44	57	2
43	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	79	19	98	74	9	88
44	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	88	53	85	88	80
45	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	4	84	6	83
46	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	77	24	64
47	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	93	98
48	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	98
49	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0

Table A.12. Distance matrix for raidl50

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
0	0	71	20	56	94	50	76	57	57	45	65	60	80	19	64	100	4	73	91	32	27	93	37	83	85	8	3	82	90	45	14	92	15	85	47	60	86	23	69	95	19	33	6	98	51	69	50	7	41	92
1	-	0	90	19	36	26	53	20	34	8	53	23	52	19	66	19	55	64	30	41	38	98	87	56	82	92	6	85	61	7	91	53	98	80	24	33	6	28	4	91	35	8	65	39	78	30	57	33	93	86
2	-	-	0	25	31	36	11	38	17	54	95	1	66	1	91	19	50	23	94	34	28	21	89	70	56	97	34	94	26	63	2	58	7	87	34	89	22	96	27	91	50	73	91	15	74	34	85	75	56	78
3	-	-	-	0	9	35	51	97	4	6	45	37	51	71	51	52	80	9	90	66	98	64	61	76	6	62	48	96	29	73	29	13	48	84	43	8	70	45	4	73	2	1	61	52	71	11	55	2	20	96
4	-	-	-	-	0	67	69	11	80	96	16	41	95	12	21	68	92	34	67	28	28	74	97	72	29	70	73	29	30	76	51	93	82	53	64	77	19	84	88	98	79	55	91	73	18	11	92	10	96	58
5	-	-	-	-	-	0	37	75	83	85	98	64	6	70	44	88	97	95	80	78	99	95	7	17	78	46	67	8	100	9	80	18	71	72	79	67	81	15	41	16	51	39	31	57	60	74	44	9	20	75
6	-	-	-	-	-	-	0	86	18	69	44	87	98	89	5	5	89	13	36	58	83	59	36	1	92	2	94	7	52	84	89	8	95	62	51	3	82	25	41	99	45	84	85	42	73	41	98	13	53	34
7	-	-	-	-	-	-	-	0	70	88	92	5	40	83	6	33	41	57	68	29	17	63	43	67	65	24	44	57	74	88	41	11	82	65	51	79	77	56	64	46	43	8	2	34	42	7	67	83	15	34
8	-	-	-	-	-	-	-	-	0	63	31	48	5	50	13	80	45	21	6	84	61	16	65	25	18	96	53	25	59	98	19	66	51	53	8	9	19	42	24	4	56	6	52	61	7	16	92	3	36	97
9	-	-	-	-	-	-	-	-	-	0	87	49	64	3	73	34	98	26	58	9	75	77	26	26	29	33	34	99	26	9	54	82	67	5	94	25	72	85	28	8	34	14	56	49	16	80	82	14	57	92
10	-	-	-	-	-	-	-	-	-	-	0	74	84	68	99	61	48	84	46	98	9	7	51	42	73	8	35	97	79	72	24	38	57	37	45	5	5	77	39	70	33	30	43	68	49	41	28	48	24	26
11	-	-	-	-	-	-	-	-	-	-	-	0	45	85	32	95	26	4	54	61	52	33	84	28	70	40	64	15	44	20	91	34	89	75	63	31	95	63	24	22	62	47	99	6	31	30	53	57	85	6
12	-	-	-	-	-	-	-	-	-	-	-	-	0	69	37	90	52	64	60	91	79	26	86	99	68	20	39	42	34	22	36	49	45	10	10	43	8	68	74	90	20	82	74	77	2	10	19	5	25	30
13	-	-	-	-	-	-	-	-	-	-	-	-	-	0	95	4	55	32	54	22	51	44	15	85	65	3	85	9	64	94	52	23	61	77	12	32	10	38	61	63	99	79	67	24	8	13	79	62	44	84
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	35	95	27	1	31	92	55	15	52	70	60	55	93	21	83	56	52	92	93	12	54	92	42	20	67	49	84	45	62	28	28	48	74	6	1
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	4	49	7	70	1	77	81	7	21	1	90	76	5	33	69	68	39	12	10	10	78	10	94	22	72	21	1	71	94	58	23	49	7	30
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	70	59	6	50	65	26	3	6	1	7	39	21	74	77	84	35	86	61	45	79	34	68	99	86	90	44	44	13	92	2	42	13	60
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	47	63	24	24	65	30	76	23	20	49	96	96	32	83	33	45	79	12	30	98	62	16	87	6	11	51	49	12	92	62	71	90
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	76	46	65	40	27	41	14	98	41	61	93	72	43	26	68	73	89	98	70	50	65	9	7	27	59	56	90	3	69	60	92
19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	44	57	57	35	36	49	48	33	89	8	78	12	3	3	80	75	43	29	45	92	45	5	99	71	63	6	60	65	74	71
20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	9	17	79	17	3	14	65	50	99	5	9	76	16	11	30	95	38	72	75	34	15	19	38	13	41	52	18	100	17
21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	43	22	25	11	1	41	13	66	57	14	64	61	75	91	76	37	20	23	74	43	49	7	58	68	96	22	60	48
22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	92	60	64	34	33	40	45	85	32	9	51	40	75	66	100	49	57	27	85	28	1	11	23	50	17	80	69
23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	65	1	28	12	44	39	27	78	72	18	74	56	49	34	58	88	8	24	39	56	32	65	93	11	66	3
24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	33	15	71	64	35	35	17	62	46	60	1	24	37	24	41	62	31	89	96	89	28	3	64	66	11
25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	47	83	3	57	48	5	42	14	75	5	48	10	21	61	7	81	13	31	69	88	23	31	19	12
26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	78	59	91	32	74	9	42	20	91	96	28	90	100	69	3	27	74	2	88	46	14	94	26
27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	27	76	95	14	99	77	84	62	6	42	4	37	67	12	31	86	2	26	66	91	78	34
28	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	45	4	59	98	43	5	12	88	82	90	64	28	55	14	4	39	27	61	32	30	50
29	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	51	42	80	88	43	57	53	86	86	39	82	89	49	80	83	53	43	71	87	84
30	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	86	14	38	99	70	28	25	30	12	6	79	62	47	10	1	42	67	54	27
31	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	52	44	60	93	92	91	75	97	85	97	83	68	82	96	58	80	65	37
32	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	56	47	100	62	77	13	60	87	14	1	5	67	79	56	10	39	100
33	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	1	81	27	49	66	23	31	33	5	79	42	36	95	79	44	41
34	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	78	57	18	91	16	56	56	69	60	22	47	67	83	37	67
35	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	35	70	45	84	35	67	66	67	23	44	61	11	91	39
36	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	54	31	68	62	100	10	77	55	65	97	66	38	96
37	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	33	20	84	51	7	53	95	90	87	13	7	6
38	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	36	51	66	98	93	56	3	23	23	64	75
39	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	33	92	81	49	89	47	87	84	31	58
40	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	67	81	64	20	27	5	58	39	12
41	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	63	26	14	80	75	6	35	77
42	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	80	58	40	54	42	32	87
43	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	90	20	85	76	55	67
44	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	86	21	99	49	92
45	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	25	54	50	15
46	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	65	64	93
47	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	30	44
48	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	67
49	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0

A.3 Berry's Test Instances

This section presents the details for the berry6, berry35, and berry35u problem instances proposed by Berry et al. (1995). Table A.13 presents the properties of the best known solutions to the different problem instances. The demands for the test problems are presented in Tables A.14(a) (berry6) and A.15(a) (berry35 and berry35u). The distance weights are shown in Tables A.14(b) (berry6) and A.15(b) (berry35). For berry35u, the distance weights $d_{ij} = 1$, for $i, j \in \{0, \ldots, 34\}$. The demands are the same for berry35 and berry35u.

	$c(T_{opt})$	used links
berry6	534	1-0, 3-1, 5-2, 5-3, 5-4
berry35u	16,273	1-0, 8-2, 11-4, 12-9, 12-10, 13-8, 15-9, 16-2, 17-3, 18-6, 19-15, 20-9, 21-8, 25-1, 25-3, 25-19, 25-24, 26-22, 27-15, 28-9, 29-8, 29-11, 29-25, 30-5, 30-14, 30-21, 30-22, 31-7, 31-12, 31-23, 32-18, 32-25, 33-25, 34-29
berry35	16,915	1-0, 8-2, 11-4, 12-9, 12-10, 13-8, 16-2, 17-3, 18-6, 19-15, 20-9, 20-15, 21-8, 24-1, 24-17, 24-18, 25-3, 25-8, 25-19, 26-22, 27-15, 28-9, 29-11, 29-25, 30-5, 30-14, 30-21, 30-22, 31-7, 31-12, 31-23, 32-18, 33-24, 34-29

Table A.13. Cost and structure of the best solutions for berry6, berry35u, and berry35

Table A.14. Demand and distance matrix for berry6

(a) Demand matrix

	0	1	2	3	4	5
0	0	5	13	12	8	9
1	-	0	7	4	2	6
2	-	-	0	3	10	15
3	-	-	-	0	11	7
4	-	-	-	-	0	12
5	-	-	-	-	-	0

(b) Distance matrix

	0	1	2	3	4	5
0	0	3	6	5	9	7
1	-	0	3	2	4	8
2	-	-	0	3	7	2
3	-	-	-	0	9	2
4	-	-	-	-	0	1
5	-	-	-	-	-	0

Table A.15. Demand and distance matrix for berry35 and berry35u

(a) Demand matrix for berry35 and berry35u

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
0	0	639	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	93	0	0	0	0	0	0	0	129	0
1	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	147	0	0	0	0	0	0	83	0	0	
2	-	-	0	0	0	0	0	0	189	0	0	0	0	0	0	0	99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	43	0	0	
3	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	531	0	0	0	0	0	0	0	0	623	0	0	0	0	0	0	0	0	
4	-	-	-	-	0	0	0	0	0	0	53	0	0	0	0	0	0	0	0	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	-	-	-	-	-	0	43	0	0	0	0	0	0	119	0	0	0	0	0	0	0	329	0	0	0	0	0	0	0	651	0	0	0	0	
6	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	371	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
7	-	-	-	-	-	-	-	0	0	0	0	0	0	0	23	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	171	0	0	
8	-	-	-	-	-	-	-	-	0	0	0	0	0	41	0	0	0	0	0	0	0	189	0	0	0	0	0	0	0	123	0	0	0	0	
9	-	-	-	-	-	-	-	-	-	0	0	0	351	0	0	0	0	0	0	0	61	0	0	0	0	0	0	0	217	0	0	0	0	0	
10	-	-	-	-	-	-	-	-	-	-	0	0	81	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	
11	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	133	0	0	0	0	
12	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	27	0	0	0	0	0	0	0	161	0	0	0	0	0	0	0	261	0	0	
13	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	261	0	0	0	0	0	0	0	639	0	0	0	0	
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	147	0	0	0	0	0	0	0	423	0	0	0	0	0	0	
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	69	0	0	0	0	0	0	0	0	0	
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	351	0	0	0	0	0	0	117	0	0	
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	243	0	0	0	0	0	0	873	0	0	
19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	639	0	0	0	0	0	0	119	0	
20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	57	0	0	0	0	0	0	0	
21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	91	0	0	0	0	0	0	0	387	0	0	0	0	
22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	89	0	0	0	0	0	0	0	0	
23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	651	0	0	0	
24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	133	0	0	0	0	0	
25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	21	0	0	0	0	0	0	
26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	
27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	111	0	0	0	0	0	
28	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	
29	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	63	
30	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	
31	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	
32	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	71	0	
33	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	
34	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	

(b) Distance matrix for berry35

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
0	0	1	7	4	8	9	4	12	6	9	11	7	10	7	9	7	8	3	3	6	8	7	10	12	2	5	9	8	10	6	8	11	4	3	7
1	-	0	6	3	7	8	3	11	5	8	10	6	9	6	8	6	7	2	2	5	7	6	9	11	1	4	8	7	9	5	7	10	3	2	6
2	-	-	0	3	5	4	7	9	1	6	8	4	7	2	4	4	1	4	6	3	5	2	5	9	5	2	4	5	7	3	3	8	7	6	4
3	-	-	-	0	4	5	4	8	2	5	7	3	6	3	5	3	4	1	3	2	4	3	6	8	2	1	5	4	6	2	4	7	4	3	3
4	-	-	-	-	0	7	8	10	4	7	9	1	8	5	7	5	6	5	7	4	6	5	8	10	6	3	7	6	8	2	6	9	8	7	3
5	-	-	-	-	-	0	9	11	3	8	10	6	9	4	2	6	5	6	8	5	7	2	3	11	7	4	2	7	9	5	1	10	9	8	6
6	-	-	-	-	-	-	0	12	6	9	11	7	10	7	9	7	8	3	1	6	8	7	10	12	2	5	9	8	10	6	8	11	2	3	7
7	-	-	-	-	-	-	-	0	8	3	3	9	2	9	11	5	10	9	11	6	4	9	12	2	10	7	11	6	4	8	10	1	12	11	9
8	-	-	-	-	-	-	-	-	0	5	7	3	6	1	3	3	2	3	5	2	4	1	4	8	4	1	3	4	6	2	2	7	6	5	3
9	-	-	-	-	-	-	-	-	-	0	2	6	1	6	8	2	7	6	8	3	1	6	9	3	7	4	8	3	1	5	7	2	9	8	6
10	-	-	-	-	-	-	-	-	-	-	0	8	1	8	10	4	9	8	10	5	3	8	11	3	9	6	10	5	3	7	9	2	11	10	8
11	-	-	-	-	-	-	-	-	-	-	-	0	7	4	6	4	5	4	6	3	5	4	7	9	5	2	6	5	7	1	5	8	7	6	2
12	-	-	-	-	-	-	-	-	-	-	-	-	0	7	9	3	8	7	9	4	2	7	10	2	8	5	9	4	2	6	8	1	10	9	7
13	-	-	-	-	-	-	-	-	-	-	-	-	-	0	4	4	3	4	6	3	5	2	5	9	5	2	4	5	7	3	3	8	7	6	4
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	6	5	6	8	5	7	2	3	11	7	4	2	7	9	5	1	10	9	8	6
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	5	4	6	1	1	4	7	5	5	2	6	1	3	3	5	4	7	6	4
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	5	7	4	6	3	6	10	6	3	5	6	8	4	4	9	8	7	5
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	2	3	5	4	7	9	1	2	6	5	7	3	5	8	3	2	4
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	5	7	6	9	11	1	4	8	7	9	5	7	10	1	2	6
19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	2	3	6	6	4	1	5	2	4	2	4	5	6	5	3
20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	5	8	4	6	3	7	2	2	4	6	3	8	7	5
21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	3	9	5	2	2	5	7	3	1	8	7	6	4
22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	12	8	5	1	8	10	6	2	11	10	9	7
23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	10	7	11	6	4	8	10	1	12	11	9
24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	3	7	6	8	4	6	9	2	1	5
25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	4	3	5	1	3	6	5	4	2
26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	7	9	5	1	10	9	8	6
27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	4	4	6	5	8	7	5
28	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	6	8	3	10	9	7
29	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	4	7	6	5	1
30	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	9	8	7	5
31	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	11	10	8
32	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	3	7
33	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	6
34	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0

A.4 Real World Problems

This section presents the properties of four real-world problems. The presented problems are no classical OCST problems as the cost of a link depends non-linearly on its distance weights d_{ij} and the traffic b_{ij} running over the link.

For fulfilling the demands between the nodes, different line types with only discrete capacities are available. The cost of installing a link consists of a fixed and length dependent share. Both depend on the capacity of the link. The costs are based on the tariffs of the German Telekom from 1996 and represent the amount of money (in German Marks) a company has to pay for a telecommunication line of a specific length and capacity per month. For a detailed description of the four different problems the reader is referred to Sect. 8.2.3. In particular, the overall cost of a communication network is calculated as

$$c(T) = \sum_{i,j \in F} f(d_{ij}, cap_{ij}),$$

where F denotes the set of used links, d_{ij} are the distance weights of the links between node i to node j, and cap_{ij} is the capacity of the links. The distance weight d_{ij} corresponds to the Euclidean distance between the nodes i and j. The capacity cap_{ij} of a link must be higher than the overall traffic b_{ij} running over a link. Therefore,

$$cap_{ij} \geq b_{ij},$$

where b_{ij} denotes the overall traffic over the link between the nodes i and j. This means that to every link between i and j a line is assigned with the next higher available capacity cap_{ij}.

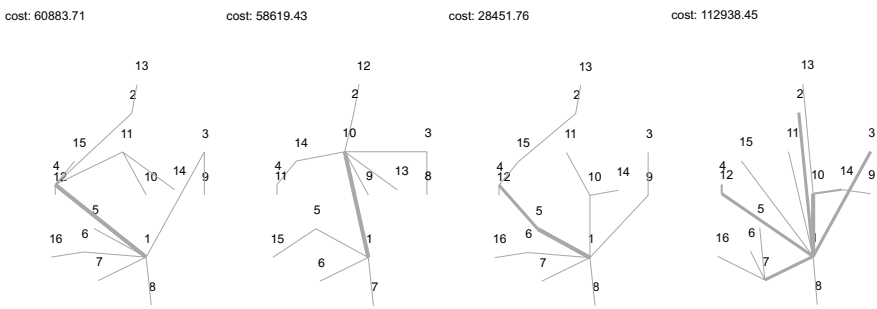
We illustrate this with a brief example. If there are three line types available with capacity 64 kBit/s, 512 kBit/s, and 2048 kBit/s, a line with capacity $cap = 64$ kBit/s is assigned to all links with less than $b = 64$ kBit/s of traffic. If the traffic over a link is between 64 kBit/s and 512 kBit/s the 512 kBit/s line is chosen. If the traffic over a link exceeds 512 kBit/s the 2048 kBit/s line must be chosen.

Table A.16 and Fig. A.1 present the properties of the best known solutions to the four test problems. Table A.17(a) (rothlauf1 and rothlauf2), Table A.17(b) (rothlauf3), and Table A.18 (rothlauf4) illustrate how the cost of a link depends on the overall traffic b_{ij} and the distance weight d_{ij} of the used line. The largest available line type has capacity $cap = 2,048$ kBit/s (rothlauf1-3) or $cap = 4,096$ kBit/s (rothlauf4). If the traffic b over a link exceeds this value a large penalty is used.

In Table A.19 (rothlauf1 and rothlauf3), Table A.20 (rothlauf2), and Table A.21 (rothlauf4) we present the demands for the different test problems. Table A.22 lists the coordinates of the nodes. The distance weights $d_{i,j}$ are calculated as $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, where x and y denote the coordinates of the nodes. To get the distances and coordinates in kilometer, the distance weights must be multiplied by 14.87. The factor 14.87 results from the used "Gebührenfeldverfahren" of the German Telekom.

	$c(T_{opt})$	used links
rothlauf1	60,883	3-1, 4-1, 4-2, 5-1, 6-1, 7-1, 8-1, 9-3, 11-4, 11-10, 12-4, 13-2, 14-11, 15-4, 16-6
rothlauf2	58,619	5-1, 6-1, 7-1, 8-3, 10-1, 10-2, 10-3, 10-9, 11-4, 12-2, 13-10, 14-4, 14-10, 15-5
rothlauf3	28,451	5-1, 5-4, 6-1, 7-1, 8-1, 9-1, 9-3, 10-1, 11-10, 12-4, 13-2, 14-10, 15-2, 15-4, 16-6
rothlauf4	112,938	2-1, 3-1, 7-1, 7-5, 7-6, 8-1, 10-1, 11-1, 12-1, 12-4, 13-10, 14-9, 14-10, 15-1, 16-7

Table A.16. Cost and structure of the best solutions for selected real-world test instances



(a) rothlauf1 (b) rothlauf2 (c) rothlauf3 (d) rothlauf4

Figure A.1. Best known solutions T_{opt} for the four real-world problem instances

Table A.17. Cost of a link for rothlauf1, rothlauf2, and rothlauf3

(a) rothlauf1 and rothlauf2

b_{ij}	d_{ij}	cost
<64 kBit/s	$[0;1]$	$334.58d + 385$
	$]1;3]$	$148.70d + 572$
	$]3;10]$	$29.74d + 972.50$
	$]10;\infty]$	$22.31d + 1,047$
<512 kBit/s	$[0;1]$	$1,107d + 975$
	$]1;3]$	$520d + 1,567$
	$]3;10]$	$178d + 2,717$
	$]10;\infty]$	$111.53d + 3,392$
<2,048 kBit/s	$[0;1]$	$2,215d + 1,950$
	$]1;3]$	$1,040.9d + 3,135$
	$]3;10]$	$356.88d + 5,435$
	$]10;\infty]$	$223.05d + 6,785$
>2,048 kBit/s	$[0;\infty]$	$500{,}000d + 50{,}000$

(b) rothlauf3

b_{ij}	d_{ij}	cost
<64 kBit/s	$[0;1]$	$334.58d + 38,5$
	$]1;3]$	$148.7d + 57,2$
	$]3;10]$	$29.74d + 97,25$
	$]10;\infty]$	$22.31d + 104.7$
<512 kBit/s	$[0;1]$	$1,107d + 97.5$
	$]1;3]$	$520d + 156.7$
	$]3;10]$	$178d + 271.7$
	$]10;\infty]$	$111.53d + 339.2$
<2,048 kBit/s	$[0;1]$	$2,215d + 195$
	$]1;3]$	$1,040.9d + 313.5$
	$]3;10]$	$356.88d + 543.5$
	$]10;\infty]$	$223.05d + 678.5$
>2,048 kBit/s	$[0;\infty]$	$500{,}000d + 50{,}000$

b_{ij}	d_{ij}	cost
<64 kBit/s	$[0;1]$	$334.58d + 385$
	$]1;3]$	$148.70d + 572$
	$]3;10]$	$29.74d + 972.50$
	$]10;\infty]$	$22.31d + 1,047$
<128 kBit/s	$[0;1]$	$669.16d + 770$
	$]1;3]$	$297.40d + 1,144$
	$]3;10]$	$59.48d + 1,945$
	$]10;\infty]$	$44.62d + 2,094$
<512 kBit/s	$[0;1]$	$1,107d + 975$
	$]1;3]$	$520d + 1,567$
	$]3;10]$	$178d + 2,717$
	$]10;\infty]$	$111.53d + 3,392$
<2,048 kBit/s	$[0;1]$	$2,215d + 1,950$
	$]1;3]$	$1,040.90d + 3,135$
	$]3;10]$	$356.88d + 5,435$
	$]10;\infty]$	$223.05d + 6,785$
<4,096 kBit/s	$[0;1]$	$4,430d + 3,900$
	$]1;3]$	$2,081.80d + 6,270$
	$]3;10]$	$713.76d + 10,870$
	$]10;\infty]$	$446.10d + 13,570$
>4,096 kBit/s	$[0;\infty]$	$500{,}000d + 50{,}000$

Table A.18. Cost of a link for rothlauf4

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
1	0	424	458	727	468	414	440	521	50	48	381	34	28	48	34	28
2	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0
5	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0
6	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0
7	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0
8	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0
9	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0
10	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0
11	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0
12	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0
13	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0
14	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0

Table A.19. Demand matrix for rothlauf1 and rothlauf3

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	424	458	200	468	440	521	50	48	600	34	28	48	34	28
2	-	0	0	0	0	0	0	0	0	0	0	0	0	40	0
3	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0
4	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0
5	-	-	-	-	0	0	0	0	0	0	0	0	0	0	100
6	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0
7	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0
8	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0
9	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0
10	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0
11	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0
12	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0
13	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0
14	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0

Table A.20. Demand matrix for rothlauf2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	308	491	364	36	51	195	72	114	111	14	150	78	136	33	44
2	-	0	503	323	27	38	146	54	86	83	11	112	59	102	24	33
3	-	-	0	272	18	25	97	36	57	55	7	75	39	68	16	22
4	-	-	-	0	9	12	48	18	28	27	3	37	19	34	8	11
5	-	-	-	-	0	51	17	1	34	40	54	36	47	45	25	11
6	-	-	-	-	-	0	15	63	22	16	31	42	28	54	33	7
7	-	-	-	-	-	-	0	5	26	62	54	45	39	12	16	18
8	-	-	-	-	-	-	-	0	32	13	40	22	20	34	61	38
9	-	-	-	-	-	-	-	-	0	35	16	54	13	38	49	17
10	-	-	-	-	-	-	-	-	-	0	10	12	47	4	5	49
11	-	-	-	-	-	-	-	-	-	-	0	49	10	55	28	39
12	-	-	-	-	-	-	-	-	-	-	-	0	10	4	48	37
13	-	-	-	-	-	-	-	-	-	-	-	-	0	19	41	38
14	-	-	-	-	-	-	-	-	-	-	-	-	-	0	17	34
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	36
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0

Table A.21. Demand matrix for rothlauf4

problem	node	x;y	node	x;y	node	x;y
rothlauf1, rothlauf3, rothlauf4	1	29;12	2	26;42	3	41;34
	4	10;27	5	18;18	6	16;13
	7	19;7	8	30;2	9	41;25
	10	29;25	11	24;34	12	10;25
	13	27;48	14	35;26	15	14;32
	16	9;12				
rothlauf2	1	29;12	2	26;42	3	41;34
	4	10;27	5	18;18	6	19;7
	7	30;2	8	41;25	9	29;25
	10	24;34	11	10;25	12	27;48
	13	35;26	14	14;32	15	9;12

Table A.22. Position of the nodes for the four selected real-world problems

References

Abramowitz, M. and I. A. Stegun (1972). *Handbook of Mathematical Functions*. New York: Dover Publications.

Abuali, F. N., D. A. Schoenefeld, and R. L. Wainwright (1994). Designing telecommunications networks using genetic algorithms and probabilistic minimum spanning trees. In E. Deaton, D. Oppenheim, J. Urban, and H. Berghel (Eds.), *Proceedings of the 1994 ACM Symposium on Applied Computing*, pp. 242–246. ACM Press.

Abuali, F. N., R. L. Wainwright, and D. A. Schoenefeld (1995). Determinant factorization: A new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem. See Eschelman (1995), pp. 470–477.

Ackley, D. H. (1987). *A connectionist machine for genetic hill climbing*. Boston: Kluwer Academic.

Albuquerque, P., B. Chopard, C. Mazza, and M. Tomassini (2000). On the impact of the representation on fitness landscapes. In R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, and T. C. Fogarty (Eds.), *Genetic Programming: Third European Conference*, Berlin, pp. 1–15. Springer-Verlag.

Alon, N., R. M. Karp, D. Peleg, and D. West (1995). A graph theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 78–100.

Altenberg, L. (1994). The schema theorem and price's theorem. See Whitley (1994), pp. 23–49.

Altenberg, L. (1997). Fitness distance correlation analysis: An instructive counterexample. In T. Bäck (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms*, San Francisco, pp. 57–64. Morgan Kaufmann.

Angeline, P. J., Z. Michalewicz, M. Schoenauer, X. Yao, A. Zalzala, and W. Porto (Eds.) (1999). *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*. IEEE Press.

Asoh, H. and H. Mühlenbein (1994). On the mean convergence time of evolutionary algorithms without selection and mutation. See Davidor, Schwefel, and Männer (1994), pp. 88–97.

Bäck, T., D. B. Fogel, and Z. Michalewicz (Eds.) (1997). *Handbook of Evolutionary Computation*. Bristol and New York: Institute of Physics Publishing and Oxford University Press.

Bäck, T. and H.-P. Schwefel (1995). Evolution strategies I: Variants and their computational implementation. In G. Winter, J. Périaux, M. Galán, and P. Cuesta (Eds.), *Genetic Algorithms in Engineering and Computer Science*, Chapter 6, pp. 111–126. Chichester: John Wiley and Sons.

Bagley, J. D. (1967). *The Behavior of Adaptive Systems Which Employ Genetic and Correlation Algorithms*. Ph. D. thesis, University of Michigan. (University Microfilms No. 68-7556).

Banzhaf, W. (1994). Genotype-phenotype-mapping and neutral variation – A case study in genetic programming. See Davidor, Schwefel, and Männer (1994), pp. 322–332.

Banzhaf, W., J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (Eds.) (1999). *Proceedings of the Genetic and Evolutionary Computation Conference: Volume 1*, San Francisco, CA. Morgan Kaufmann Publishers.

Barnett, L. (1997). Tangled webs: Evolutionary dynamics on fitness landscapes with neutrality. Master's thesis, School of Cognitive Sciences, University of East Sussex, Brighton.

Barnett, L. (1998, June 27–29). Ruggedness and neutrality: The NKp family of fitness landscapes. In C. Adami, R. K. Belew, H. Kitano, and C. Taylor (Eds.), *Proceedings of the 6th International Conference on Artificial Life (ALIFE-98)*, Cambridge, MA, USA, pp. 18–27. MIT Press.

Barnett, L. (2001). Netcrawling - optimal evolutionary search with neutral networks. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC01*, Piscataway, NJ, pp. 30–37. IEEE Press.

Bartal, Y. (1996). Probabilistic approximation of metric spaces and its algorithmic applications. In *Proc. 37th IEEE Symp. on Foundations of Computer Science*, pp. 184–193.

Bartal, Y. (1998). On approximating arbitrary metrics by tree metrics. In *Proc. 30th Anual ACM Symp. on Theory of Computer Science*, pp. 161–168.

Bean, J. C. (1992, June). Genetics and random keys for sequencing and optimization. Technical Report 92-43, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing 6*(2), 154–160.

Beasley, D., D. R. Bull, and R. R. Martin (1993). Reducing epitasis in combinatorial problems by expansive coding. See Forrest (1993), pp. 400–407.

Belew, R. K. and L. B. Booker (Eds.) (1991). *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA. Morgan Kaufmann.

Berry, L. T. M., B. A. Murtagh, and G. McMahon (1995). Applications of a genetic-based algorithm for optimal design of tree-structured communication networks. In *Proceedings of the Regional Teletraffic Engineering Conference of the International Teletraffic Congress*, Pretoria, South Africa, pp. 361–370.

Berry, L. T. M., B. A. Murtagh, G. McMahon, and S. Sugden (1997). Optimization models for communication network design. In *Proceedings of the Fourth International Meeting Decision Sciences Institute*, Sydney, Australia, pp. 67–70.

Berry, L. T. M., B. A. Murtagh, G. McMahon, S. Sugden, and L. Welling (1999). An integrated GA–LP approach to communication network design. *Telecommunication Systems 12*(2), 265–280.

Berry, L. T. M., B. A. Murtagh, and S. J. Sugden (1994). A genetic-based approach to tree network synthesis with cost constraints. In H. J. Zimmermann (Ed.), *Second European Congress on Intelligent Techniques and Soft Computing - EUFIT'94*, Volume 2, Promenade 9, D-52076 Aachen, pp. 626–629. Verlag der Augustinus Buchhandlung.

Bethke, A. D. (1981). *Genetic Algorithms as Function Optimizers*. Ph. D. thesis, University of Michigan. (University Microfilms No. 8106101).

Bosman, P. (2003). *Design and Application of Iterated Density-Estimation Evolutionary Algorithms*. Ph. D. thesis, Universiteit Utrecht, Utrecht, The Netherlands.

Brittain, D. (1999). *Optimisation of the telecommunications access network*. Unpublished doctoral dissertation, University of Bristol, Bristol.

Brittain, D., J. S. Williams, and C. McMahon (1997). A genetic algorithm approach to planning the telecommunications access network. In T. Bäck (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms*, San Francisco, pp. 623–628. Morgan Kaufmann.

Cahn, R. S. (1998). *Wide Area Network Design, Concepts and Tools for Optimization*. San Francisco: Morgan Kaufmann Publishers.

Caruana, R. A. and J. D. Schaffer (1988). Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In L. Laird (Ed.), *Proceedings of the Fifth International Workshop on Machine Learning*, San Mateo, CA, pp. 153–161. Morgan Kaufmann.

Caruana, R. A., J. D. Schaffer, and L. J. Eshelman (1989). Using multiple representations to improve inductive bias: Gray and binary coding for genetic algorithms. In B. Spatz (Ed.), *Proceedings of the Sixth International Workshop on Machine Learning*, San Mateo, CA, pp. 375–378. Morgan Kaufmann.

Cavicchio, Jr., D. J. (1970). *Adaptive Search Using Simulated Evolution*. Unpublished doctoral dissertation, University of Michigan, Ann Arbor, MI. (University Microfilms No. 25-0199).

Cayley, A. (1889). A theorem on trees. *Quarterly Journal of Mathematics 23*, 376–378.

Celli, G., E. Costamagna, and A. Fanni (1995, October). Genetic algorithm for telecommunication network optimization. In *IEEE Int. Conf. on Systems, Man and Cybernetics*, Volume 2, Vancouver, pp. 1227–1232. IEEE Systems, Man and Cybernetics Society.

Charikar, M., C. Chekuri, A. Goel, S. Guha, and S. Plotkin (1998, November). Approximating a finite metric by a small number of tree metrics. In *Proc. 39th IEEE Symp. on Foundations of Computer Science*, pp. 111–125.

Chou, H., G. Premkumar, and C.-H. Chu (2001, June). Genetic algorithms for communications network design - an empirical study of the factors that influence performance. *IEEE Transactions on Evolutionary Computation 5*(3), 236–249.

Christensen, S. and F. Oppacher (2001). What can we learn from no free lunch? See Spector, E., Wu, B., Voigt, Gen, Sen, Dorigo, Pezeshk, Garzon, and Burke (2001), pp. 1219–1226.

Chu, C.-H., C. Chou, and G. Premkumar (2000). Digital data networks design using genetic algorithms. *European Journal of Operational Research 127*(1), 140–158.

Chu, C.-H., G. Premkumar, C. Chou, and J. Sun (1999). Dynamic degree constrained network design: A genetic algorithm approach. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999), pp. 141–148.

Chu, C. H.and Chou, H. and G. Premkumar (1999). Digital data networks design using genetic algorithms. Technical Report 07-1999, Penn State, USA.

Cohoon, J. P., S. U. Hegde, W. N. Martin, and D. Richards (1988). Floorplan design using distributed genetic algorithms. In *IEEE International Conference on Computer Aided-Design*, pp. 452–455. IEEE.

Coli, M. and P. Palazzari (1995a). Searching for the optimal coding in genetic algorithms. In *1995 IEEE International Conference on Evolutionary Computation*, Volume 1, Piscataway, NJ, pp. 92–96. IEEE Service Center.

Coli, M. and P. Palazzari (1995b). Searching for the optimal coding in genetic algorithms. *1995 IEEE International Conference on Evolutionary Computation 1*, 92–96.

Crescenzi, P. and V. Kann (2003, Aug.). A compendium of NP optimization problems. `http://www.nada.kth.se/theory/compendium`.

Darwin, C. (1859). *On the Origin of Species*. London: John Murray.

Dasgupta, D. (1995). Incorporating redundancy and gene activation mechanisms in genetic search for adapting to non-stationary environments. In L. Chambers (Ed.), *Practical Handbook of Genetic Algorithms*, Chapter 13, pp. 303–316. CRC Press.

Davidor, Y. (1989). Epistasis variance – Suitability of a representation to genetic algorithms. Tech. Rep. No. CS89-25, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel.

Davidor, Y. (1991). Epistasis variance: A viewpoint on GA-hardness. See Rawlins (1991), pp. 23–35.

Davidor, Y., H.-P. Schwefel, and R. Männer (Eds.) (1994). *Parallel Problem Solving from Nature- PPSN III*, Berlin. Springer-Verlag.

Davis, L. (1987). *Genetic Algorithms and Simulated Annealing*. San Mateo, CA: Morgan Kaufmann.

Davis, L. (1989). Adapting operator probabilities in genetic algorithms. See Schaffer (1989), pp. 61–69.

Davis, L., D. Orvosh, A. Cox, and Y. Qiu (1993). A genetic algorithm for survivable network design. See Forrest (1993), pp. 408–415.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Ph. D. thesis, University of Michigan, Ann Arbor. (University Microfilms No. 76-9381).

Deb, K., L. Altenberg, B. Manderick, T. Bäck, Z. Michalewicz, M. Mitchell, and S. Forrest (1997). Theoretical foundations and properties of evolutionary computations: fitness landscapes. See Bäck, Fogel, and Michalewicz (1997), pp. B2.7:1–B2.7:25.

Deb, K. and D. E. Goldberg (1993). Analyzing deception in trap functions. See Whitley (1993), pp. 93–108.

Deb, K. and D. E. Goldberg (1994). Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence 10*, 385–408.

Dengiz, B., F. Altiparmak, and A. E. Smith (1997a). Efficient optimization of all-terminal reliable networds, using an evolutionary approach. *IEEE Transactions on Reliability 46*(1), 18–26.

Dengiz, B., F. Altiparmak, and A. E. Smith (1997b). Local search genetic algorithm for optimal design of reliable networks. *IEEE Transactions on Evolutionary Computation 1*(3), 179–188.

Dengiz, B., F. Altiparmak, and A. E. Smith (1997c). Local search genetic algorithm for optimization of highly reliable communications networks. In T. Bäck (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms*, San Francisco, pp. 650–657. Morgan Kaufmann.

Dionne, R. and M. Florian (1979). Exact and approximate algorithms for optimal network design. *Networks 9*, 39–59.

Doran, J. and D. Michie (1966). Experiments with the graph traverser program. *Proceedings of the Royal Society of London (A) 294*, 235–259.

Ebner, M., P. Langguth, J. Albert, M. Shackleton, and R. Shipman (2001, 27-30 May). On neutral networks and evolvability. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, pp. 1–8. IEEE Press.

Edelson, W. and M. L. Gargano (2000). Feasible encodings for GA solutions of constrained minimal spanning tree problems. See Whitley, Goldberg, Cantú-Paz, Spector, Parmee, and Beyer (2000), pp. 754.

Edelson, W. and M. L. Gargano (2001, 7 July). Leaf constrained minimal spanning trees solved by a GA with feasible encodings. In A. S. Wu (Ed.), *Proceedings of the 2001 Genetic and Evolutionary Computaton Conference Workshop Program*, San Francisco, California, USA, pp. 268–271.

Elbaum, R. and M. Sidi (1996). Topological design of local-area networks using genetic algorithms. *IEEE/ACM Transactions on Networking 4*(5), 766–778.

Eschelman, L. (Ed.) (1995). *Proceedings of the Sixth International Conference on Genetic Algorithms*, San Francisco, CA. Morgan Kaufmann.

Eshelman, L. J. and J. D. Schaffer (1991). Preventing premature convergence in genetic algorithms by preventing incest. See Belew and Booker (1991), pp. 115–122.

Even, S. (1973). *Algorithmic Combinatorics*. New York: The Macmillan Company.

Feller, W. (1957). *An Introduction to Probability Theory and its Applications* (1st ed.), Volume 1. New York: John Wiley & Sons.

Forrest, S. (Ed.) (1993). *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, CA. Morgan Kaufmann.

Fox, B. R. and M. B. McMahon (1991). Genetic operators for sequencing problems. See Rawlins (1991), pp. 284–300.

Gale, J. S. (1990). *Theoretical Population Genetics*. London: Unwin Hyman.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman.

Gargano, M. L., W. Edelson, and O. Koval (1998). A genetic algorithm with feasible search space for minimal spanning trees with time-dependent edge costs. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo (Eds.), *Genetic Programming 98*, San Francisco, pp. 495. Morgan Kaufmann Publishers.

Gaube, T. (2000, Februar). Optimierung baumförmiger Netzwerkstrukturen mit Hilfe des Link and Node Biased Encodings. Master's thesis, Universität Bayreuth, Lehrstuhl für Wirtschaftsinformatik.

Gaube, T. and F. Rothlauf (2001). The link and node biased encoding revisited: Bias and adjustment of parameters. In E. J. W. Boers, S. Cagnoni, J. Gottlieb, E. Hart, P. L. Lanzi, G. R. Raidl, R. E. Smith, and H. Tijink (Eds.), *Applications of Eolutionary Computing: Proc. EvoWorkshops 2001*, Berlin, pp. 1–10. Springer.

Gavish, B. (1983). Formulations and algorithms for the capacitated minimal directed tree problem. *Journal of the ACM 30*(1), 118–132.

Gavish, B. and K. Altinkemer (1990). Backbone network design tools with economic tradeoffs. *ORSA Journal on Computing 2*(3), 58–76.

Gen, M., K. Ida, and J. Kim (1998). A spanning tree-based genetic algorithm for bicriteria topological network design. See Institute of Electrical and Electronics Engineers (1998), pp. 15–20.

Gen, M. and Y. Li (1999). Spanning tree-based genetic algorithms for the bicriteria fixed charge transportation problem. See Angeline, Michalewicz, Schoenauer, Yao, Zalzala, and Porto (1999), pp. 2265–2271.

Gen, M., G. Zhou, and M. Takayama (1998). A comparative study of tree encodings on spanning tree problems. See Institute of Electrical and Electronics Engineers (1998), pp. 33–38.

Gerrits, M. and P. Hogeweg (1991). Redundant coding of an NP-complete problem allows effective Genetic Algorithm search. In H.-P. Schwefel and R. Männer (Eds.), *Parallel Problem Solving from Nature*, Berlin, pp. 70–74. Springer-Verlag.

Gerstacker, J. (1999, Februar). Netzwerkplanung durch Einsatz naturanaloger Verfahren. Master's thesis, Universität Bayreuth, Lehrstuhl für Wirtschaftsinformatik.

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal, deceptive problem. See Davis (1987), Chapter 6, pp. 74–88.

Goldberg, D. E. (1989a). Genetic algorithms and Walsh functions: Part I, a gentle introduction. *Complex Systems 3*(2), 129–152.

Goldberg, D. E. (1989b). Genetic algorithms and Walsh functions: Part II, deception and its analysis. *Complex Systems 3*(2), 153–171.

Goldberg, D. E. (1989c). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Goldberg, D. E. (1990a). A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems 4*(4), 445–460. (Also IlliGAL Report No. 90003).

Goldberg, D. E. (1990b, September). Real-coded genetic algorithms, virtual alphabets, and blocking. IlliGAL Report No. 90001, University of Illinois at Urbana-Champaign, Urbana, IL.

Goldberg, D. E. (1991a). Genetic algorithm theory. Fourth International Conference conference on Genetic Algorithms Tutorial, unpublished manuscript.

Goldberg, D. E. (1991b). Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems 5*(2), 139–167. (Also IlliGAL Report 90001).

Goldberg, D. E. (1992). Construction of high-order deceptive functions using low-order Walsh coefficients. *Annals of Mathematics and Artificial Intelligence 5*, 35–48.

Goldberg, D. E. (1998). The race, the hurdle, and the sweet spot: Lessons from the genetic algorithms for the automation of design innovation and creativity. IlliGAL Report No. 98007, University of Illinois at Urbana-Champaign, Urbana, IL.

Goldberg, D. E. (1999). The race, the hurdle, and the sweet spot. In P. J. Bentley (Ed.), *Evolutionary Design by Computers*, pp. 105–118. San Francisco, CA: Morgan Kaufmann.

Goldberg, D. E. (2002). *The Design of Innovation*. Series on Genetic Algorithms and Evolutionary Computation. Dordrecht, The Netherlands: Kluwer.

Goldberg, D. E., K. Deb, and J. H. Clark (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems 6*, 333–362.

Goldberg, D. E., K. Deb, H. Kargupta, and G. Harik (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. See Forrest (1993), pp. 56–64.

Goldberg, D. E., K. Deb, and D. Thierens (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instrument and Control Engineers 32*(1), 10–16.

Goldberg, D. E., B. Korb, and K. Deb (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems 3*(5), 493–530.

Goldberg, D. E. and P. Segrest (1987). Finite Markov chain analysis of genetic algorithms. In J. J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, Hillsdale, NJ, pp. 1–8. Lawrence Erlbaum Associates.

Gomory, R. E. and T. C. Hu (1961). Multi-terminal network flows. In *SIAM Journal on Applied Math*, Volume 9, pp. 551–570.

Gottlieb, J. and C. Eckert (2000). A comparison of two representations for the fixed charge transportation problem. See Schoenauer, Deb, Rudolph, Yao, Lutton, Merelo, and Schwefel (2000), pp. 345–354.

Gottlieb, J., B. A. Julstrom, G. R. Raidl, and F. Rothlauf (2001). Prüfer numbers: A poor representation of spanning trees for evolutionary search. See Spector, E., Wu, B., Voigt, Gen, Sen, Dorigo, Pezeshk, Garzon, and Burke (2001), pp. 343–350.

Gottlieb, J. and G. R. Raidl (2000). The effects of locality on the dynamics of decoder-based evolutionary search. See Whitley, Goldberg, Cantú-Paz, Spector, Parmee, and Beyer (2000), pp. 283–290.

Gottlieb, J. and G. Raidl (1999). Characterizing locality in decoder-based eas for the multidimensional knapsack problem. In C. Fonlupt, J.-K. Hao, E. Lutton, E. Ronald, and M. Schoenauer (Eds.), *Proceedings of Artificial Evolution*, Volume 1829 of *Lecture Notes in Computer Science*, pp. 38–52. Springer.

Grasser, C. (2000). Multiperiodenplanung von Kommunikationsnetzwerken mit naturanalogen Verfahren. Master's thesis, Universität Bayreuth, Lehrstuhl für Wirtschaftsinformatik.

Güls, D. (1996). Optimierung der Netzkonfiguration eines Corporate network am Beispiel des DATEV-Genossenschaftsnetzes. Master's thesis, Universität Bayreuth, Lehrstuhl für Wirtschaftsinformatik.

Hamming, R. (1980). *Coding and Information Theory*. Prentice-Hall.

Harik, G. (1999). Linkage learning via probabilistic modeling in the ECGA. IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Urbana, IL.

Harik, G., E. Cantú-Paz, D. E. Goldberg, and B. L. Miller (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation 7*(3), 231–253.

Harik, G. R., E. Cantú-Paz, D. E. Goldberg, and B. L. Miller (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In T. Bäck (Ed.), *Proceedings of the Forth International Conference on Evolutionary Computation*, New York, pp. 7–12. IEEE Press.

Harik, G. R. and D. E. Goldberg (1996). Learning linkage. In R. K. Belew and M. D. Vose (Eds.), *Foundations of Genetic Algorithms 4*, San Francisco, CA, pp. 247–262. Morgan Kaufmann.

Hartl, D. L. and A. G. Clark (1997). *Principles of population genetics* (3 ed.). Sunderland, Massachusetts: Sinauer Associates.

Harvey, L. and A. Thompson (1997). Through the labyrinth evolution finds a way: A silicon ridge. *Lecture Notes in Computer Science 1259*, 406–422.

Hinterding, R. (2000, 6-9 July). Representation, mutation and crossover issues in evolutionary computation. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, La Jolla Marriott Hotel La Jolla, California, USA, pp. 916–923. IEEE Press.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Horn, J. (1995). Genetic algorithms, problem difficulty, and the modality of fitness landscapes. IlliGAL Report No. 95004, University of Illinois at Urbana-Champaign, Urbana, IL.

Hu, T. C. (1974, September). Optimum communication spanning trees. *SIAM Journal on Computing 3*(3), 188–195.

Huynen, M. (1996). Exploring phenotype space through neutral evolution. *J. Mol. Evol. 43*, 165–169.

Huynen, M., P. Stadler, and W. Fontana (1996). Smoothness within ruggedness: The role of neutrality in adaptation. In *Proceedings of the National Academy of Sciences of the USA, 1993*, Washington, D.C., pp. 397–401. National Academy of Sciences.

Igel, C. (1998). Causality of hierarchical variable length representations. See Institute of Electrical and Electronics Engineers (1998), pp. 324–329.

Institute of Electrical and Electronics Engineers (Ed.) (1998). *Proceedings of 1998 IEEE International Conference on Evolutionary Computation*, Piscataway, NJ. IEEE Service Center.

Johnson, D. S., J. K. Lenstra, and A. H. G. R. Kan (1978). The complexity of the network design problem. *Networks 8*, 279–285.

Jones, T. (1995). *Evolutionary algorithms and heuristic search*. Unpublished doctoral dissertation, University of New Mexico, Alberquerque, NM.

Jones, T. and S. Forrest (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. See Eschelman (1995), pp. 184–192.

Julstrom, B. A. (1993). A genetic algorithm for the rectilinear steiner problem. See Forrest (1993), pp. 474–480.

Julstrom, B. A. (1999). Redundant genetic encodings may not be harmful. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999), pp. 791.

Julstrom, B. A. (2000). Comparing lists of edges with two other genetic codings of rectilinear steiner trees. In *Late Breaking Papers at the 2000 Genetic And Evolutionary Computation Conference*, Madison, WI, pp. 155–161. Omni Press.

Julstrom, B. A. (2001). The blob code: A better string coding of spanning trees for evolutionary search. In A. S. Wu (Ed.), *Proceedings of the 2001 Genetic and Evolutionary Computaton Conference Workshop Program*, San Francisco, California, USA, pp. 256–261.

Julstrom, B. A. (2005). The blob code is competitive with edge-sets in genetic algorithms for the minimum routing cost spanning tree problem. In Beyer, Hans-Georg et al. (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference 2005*, New York, pp. 585–590. ACM Press.

Kargupta, H. (2000a). The genetic code and the genome representation. In A. S. Wu (Ed.), *Proceedings of the 2000 Genetic and Evolutionary Computaton Conference Workshop Program*, Las Vegas, Nevada, pp. 179–184.

Kargupta, H. (2000b). The genetic code-like transformations and their effect on learning functions. See Schoenauer, Deb, Rudolph, Yao, Lutton, Merelo, and Schwefel (2000), pp. 99–108.

Kargupta, H. (2001). A striking property of genetic code-like transformations. *Complex Systems 13*(1), 1–32.

Kargupta, H., K. Deb, and D. E. Goldberg (1992). Ordering genetic algorithms and deception. See Männer and Manderick (1992), pp. 47–56.

Kershenbaum, A. (1993). *Telecommunications network design algorithms*. New York: McGraw Hill.

Kim, J. R. and M. Gen (1999). Genetic algorithm for solving bicriteria network topology design problem. See Angeline, Michalewicz, Schoenauer, Yao, Zalzala, and Porto (1999), pp. 2272–2279.

Kimura, M. (1962). On the probability of fixation of mutant genes in a population population. *Genetics 47*, 713–719.

Kimura, M. (1964). Diffusion models in population genetics. *J. Appl. Prob. 1*, 177–232.

Kimura, M. (1983). *The Neutral Theory of Molecular Evolution*. Cambridge University Press.

Knjazew, D. (2000). Application of the fast messy genetic algorithm to permutation and scheduling problems. IlliGAL Report No. 2000022, University of Illinois at Urbana-Champaign, Urbana, IL.

Knjazew, D. and D. E. Goldberg (2000). Large-scale permutation optimization with the ordering messy genetic algorithm. IlliGAL Report No. 2000013, University of Illinois at Urbana-Champaign, Urbana, IL.

Knowles, J., D. Corne, and M. Oates (1999). A new evolutionary approach to the degree constrained minimum spanning tree problem. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999), pp. 794.

Knowles, J. D. and R. A. Watson (2002). On the utility of redundant encodings in mutation-based evolutionary search. In J. J. Merelo, P. Adamidis, H.-G. Beyer, J.-L. Fernandez-Villacanas, and H.-P. Schwefel (Eds.), *Parallel Problem Solving from Nature, PPSN VII*, Berlin, pp. 88–98. Springer-Verlag.

Ko, K.-T., K.-S. Tang, C.-Y. Chan, K.-F. Man, and S. Kwong (1997, August). Using genetic algorithms to design mesh networks. *Computer 30*(8), 56–61.

Krishnamoorthy, M. and A. T. Ernst (2001). Comparison of algorithms for the degree constrained minimum spanning tree. *Journal of Heuristics 7*, 587–611.

Krishnamoorthy, M., A. T. Ernst, and Y. M. Sharaiha (1999). Comparison of algorithms for the degree constrained minimum spanning tree. Tech. rep., CSIRO Mathematical and Information Sciences, Clayton, Australia.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the travelling salesman problem. *Proc. Amer. Math. Soc.* (7), 48–50.

Larranaga, P., R. Etxeberria, J. A. Lozano, and J. M. Peña (1999, December). Optimization by learning and simulation of bayesian and gaussian networks. Technical report, Intelligent Systems Group, Dept. of Computer Science and Artificial Intelligence, University of the Basque Country. EHU-KZAA-IK-4/99.

Lewontin, R. C. (1974). *The Genetic Basis of Evolutionary Change*. Number 25 in Columbia biological series. New York: Columbia University Press.

Li, Y. (2001). An effective implementation of a direct spanning tree representation in GAs. In E. J. W. Boers, S. Cagnoni, J. Gottlieb, E. Hart, P. L. Lanzi, G. R. Raidl, R. E. Smith, and H. Tijink (Eds.), *Applications of evolutionary Computing: Proc. EvoWorkshops 2001*, Berlin, pp. 11–19. Springer.

Li, Y. and Y. Bouchebaba (1999). A new genetic algorithm for the optimal communication spanning tree problem. In C. Fonlupt, J.-K. Hao, E. Lutton, E. Ronald, and M. Schoenauer (Eds.), *Proceedings of Artificial Evolution: Fifth European Conference*, Berlin, pp. 162–173. Springer.

Li, Y., M. Gen, and K. Ida (1998). Fixed charge transportation problem by spanning tree-based genetic algorithm. *Beijing Mathematics 4*(2), 239–249.

Liepins, G. E. and M. D. Vose (1990). Representational issues in genetic optimization. *Journal of Experimental and Theoretical Artificial Intelligence 2*, 101–115.

Liepins, G. E. and M. D. Vose (1991). Polynomials, basis sets, and deceptiveness in genetic algorithms. *Complex Systems 5*(1), 45–61.

Lin, S. (1982). Effective use of heuristic algorithms in network design. In *Proceedings of Symposia in Applied Mathematics*, Volume 26, pp. 63–84.

Lobo, F. G., D. E. Goldberg, and M. Pelikan (2000). Time complexity of genetic algorithms on exponentially scaled problems. See Whitley, Goldberg, Cantú-Paz, Spector, Parmee, and Beyer (2000), pp. 151–158.

Mahfoud, S. W. and D. E. Goldberg (1995). Parallel recombinative simulated annealing: A genetic algorithm. In *Parallel Computing*, Volume 21, pp. 1–28. Amsterdam, The Netherlands: Elsevier Science.

Manderick, B., M. de Weger, and P. Spiessens (1991). The genetic algorithm and the structure of the fitness landscape. See Belew and Booker (1991), pp. 143–150.

Männer, R. and B. Manderick (Eds.) (1992). *Parallel Problem Solving from Nature- PPSN II*, Berlin. Springer-Verlag.

Martin, W. and W. Spears (Eds.) (2000). *Foundations of Genetic Algorithms 6*, San Francisco, CA. Morgan Kaufmann.

Mason, A. (1995). A non-linearity measure of a problem's crossover suitability. In *1995 IEEE International Conference on Evolutionary Computation*, Volume 1, Piscataway, NJ, pp. 68–73. IEEE Service Center.

Mayr, E. (1991). *One Long Argument: Charles Darwin and the Genesis of Modern Evolutionary Thought*. Cambridge, Massachusetts: Harvard University Press.

Mendel, G. (1866). Versuche über Pflanzen-Hybriden. In *Verhandlungen des naturforschenden Vereins*, Volume 4, Brünn, pp. 3–47. Naturforschender Verein zu Brünn.

Miller, B. L. (1997). *Noise, sampling, and efficient genetic algorithms*. doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana. Also IlliGAL Report No. 97001.

Miller, B. L. and D. E. Goldberg (1996a). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation 4*(2), 113–131.

Miller, B. L. and D. E. Goldberg (1996b). Optimal sampling for genetic algorithms. In C. H. Dagli, M. Akay, C. L. P. Chen, B. R. Fernández, and J. Ghosh (Eds.), *Proceedings of the Artificial Neural Networks in Engineering (ANNIE '96) conference*, Volume 6, New York, pp. 291–297. ASME Press.

Minoux, M. (1987). Network synthesis and dynamic network optimization. *Ann. Discrete Math. 31*, 283–323.

Mühlenbein, H. and T. Mahnig (1999). FDA - a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation 7*(4), 353–376.

Mühlenbein, H. and G. Paaß (1996). From recombination of genes to the estimation of distributions i. binary parameters. See Voigt, Ebeling, Rechenberg, and Schwefel (1996), pp. 178–187.

Mühlenbein, H. and D. Schlierkamp-Voosen (1993). Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation 1*(1), 25–49.

Nagylaki, T. (1992). *Introduction to Theoretical Population Genetics*, Volume 21 of *Biomathematics*. Berlin: Springer-Verlag.

Naudts, B., D. Suys, and A. Verschoren (1997). Epistasis as a basic concept in formal landscape analysis. In T. Bäck (Ed.), *Proceedings of the Seventh*

International Conference on Genetic Algorithms, San Francisco, pp. 65–72. Morgan Kaufmann.

Norman, B. A. (1995). *Scheduling Using the Random Keys Genetic Algorithm*. unpublished PhD thesis, University of Michigan, Ann Arbor, Michigan.

Norman, B. A. and J. C. Bean (1994). Random keys genetic algorithm for job shop scheduling. Tech. Rep. No. 94-5, The University of Michigan, Ann Arbor, MI.

Norman, B. A. and J. C. Bean (1997). Operation sequencing and tool assignment for multiple spindle CNC machines. In *Proceedings of the Forth International Conference on Evolutionary Computation*, Piscataway, NJ, pp. 425–430. IEEE.

Norman, B. A. and J. C. Bean (2000). Scheduling operations on parallel machines. *IIE Transactions 32*(5), 449–459.

Norman, B. A. and A. E. Smith (1997). Random keys genetic algorithm with adaptive penalty function for optimization of constrained facility layout problems. In *Proceedings of the Forth International Conference on Evolutionary Computation*, Piscataway, NJ, pp. 407–411. IEEE.

Norman, B. A., A. E. Smith, and R. A. Arapoglu (1998). Integrated facility design using an evolutionary approach with a subordinate network algorithm. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel (Eds.), *Parallel Problem Solving from Nature, PPSN V*, Berlin, pp. 937–946. Springer-Verlag.

Oei, C. K. (1992). Walsh function analysis of genetic algorithms of non-binary strings. Master's thesis, University of Illinois at Urbana-Champaign, Department of Computer Science, Urbana.

Orvosh, D. and L. Davis (1993). Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints. See Forrest (1993), pp. 650.

Palmer, C. C. (1994). *An approach to a problem in network design using genetic algorithms*. unpublished PhD thesis, Polytechnic University, Troy, NY.

Palmer, C. C. and A. Kershenbaum (1994a). Representing trees in genetic algorithms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, Volume 1, Piscataway, NJ, pp. 379–384. IEEE Service Center.

Palmer, C. C. and A. Kershenbaum (1994b). Two algorithms for finding optimal communication spanning trees. IBM research report RC-19394.

Papadimitriou, C. H. and M. Yannakakis (1991). Optimization, approximation, and complexity classes. *J. Comput. System Sci. 43*, 425–440.

Peleg, D. (1997). Approximating minimum communication spanning trees. Proc. 4th Colloq. on Structural Information and Communication Complexity, Ascona, Switzerland.

Peleg, D. and E. Reshef (1998). Deterministic polylog approximation for minimum communication spanning trees. *Lecture Notes in Computer Science 1443*, 670–682.

Pelikan, M. (2002). *Bayesian Optimization Algorithm: From Single Level to Hierarchy*. Ph. D. thesis, University of Illinois at Urbana-Champaign.

Pelikan, M., D. E. Goldberg, and E. Cantú-Paz (1999). BOA: The Bayesian optimization algorithm. IlliGAL Report No. 99003, University of Illinois at Urbana-Champaign, Urbana, IL.

Pelikan, M., D. E. Goldberg, and F. Lobo (1999). A survey of optimization by building and using probabilistic models. IlliGAL Report No. 99018, University of Illinois at Urbana-Champaign, Urbana, IL.

Picciotto, S. (1999). *How to encode a tree*. Ph. D. thesis, University of California, San Diego, USA.

Piggott, P. and F. Suraweera (1993). Encoding graphs for genetic algorithms: An investigation using the minimum spanning tree problem. In X. Yao (Ed.), *Preprints of the AI'93 Workshop on Evolutionary Computation*, pp. 37–48. Canberra, Australia: University of New South Wales, Australian Defense Force Academy.

Poli, R. (2001a). General schema theory for genetic programming with subtree-swapping crossover. In J. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, A. G. B. Tetamanzi, and W. B. Langdon (Eds.), *Proceedings of the Fourth European Conference on Genetic Programming (EuroGP-2001)*, Volume 2038 of *LNCS*, Lake Como, Italy, pp. 143–159. Springer Verlag.

Poli, R. (2001b). Recursive conditional schema theorem, convergence an population sizing in genetic algorithms. See Martin and Spears (2000), pp. 143–163.

Premkumar, G., C. Chu, and H. Chou (2001). Telecommunications network design decision - a genetic algorithm approach. *Decision Sciences 31*(2), 483–506.

Price, G. R. (1970). Selection and covariance. *Nature 227*, 520–521.

Prim, R. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal 36*, 1389–1401.

Prüfer, H. (1918). Neuer Beweis eines Satzes über Permutationen. *Archiv für Mathematik und Physik 27*, 742–744.

Radcliffe, N. J. (1991a). Equivalence class analysis of genetic algorithm. *Complex Systems 5*(2), 183–205.

Radcliffe, N. J. (1991b). Forma analysis and random respectful recombination. See Rawlins (1991), pp. 222–229.

Radcliffe, N. J. (1992). Non-linear genetic representations. See Männer and Manderick (1992), pp. 259–268.

Radcliffe, N. J. (1993a). Genetic set recombination. See Whitley (1993), pp. 203–219.

Radcliffe, N. J. (1993b). Genetic set recombination. *Foundations of Genetic Algorithms 2*, 203–219.

Radcliffe, N. J. (1994). The algebra of genetic algorithms. *Annals of Maths. and Artificial Intelligence 10*, 339–384.

Radcliffe, N. J. (1997). Theoretical foundations and properties of evolutionary computations: schema processing. See Bäck, Fogel, and Michalewicz (1997), pp. B2.5:1–B2.5:10.

Radcliffe, N. J. and P. D. Surry (1994). Formae and the variance of fitness. See Whitley (1994), pp. 51–72.

Raidl, G. R. (2000). An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. In *Proceedings of 2000 IEEE International Conference on Evolutionary Computation*, Piscataway, NJ, pp. 43–48. IEEE.

Raidl, G. R. (2001, February). Various instances of optimal communication spanning tree problems. personal communciation.

Raidl, G. R. and C. Drexel (2000, 8 July). A predecessor coding in an EA for the capacitated minimum spanning tree problem. In D. Whitley (Ed.), *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, Las Vegas, Nevada, USA, pp. 309–316.

Raidl, G. R. and B. A. Julstrom (2000). A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim (Eds.), *Proceedings of the 2000 ACM Symposium on Applied Computing*, pp. 440–445. ACM Press.

Raidl, G. R. and B. A. Julstrom (2003). Edge-sets: An effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation 7*(3), 225–239.

Rana, S. and W. Whitley (1998). Search, binary representations, and counting optima. In *Proceeding of a workshop on Evolutionary Algorithms. Sponsored by the Institute for Mathematics and its Applications*, pp. 1–11. Colorado State University.

Rana, S. B. and L. D. Whitley (1997). Bit representations with a twist. In T. Bäck (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms*, San Francisco, pp. 188–195. Morgan Kaufmann.

Rawlins, G. J. E. (Ed.) (1991). *Foundations of Genetic Algorithms*, San Mateo, CA. Morgan Kaufmann.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart-Bad Cannstatt: Friedrich Frommann Verlag.

Reeves, C. (2000). Fitness landscapes: A guided tour. Joint tutorials of SAB 2000 and PPSN 2000, tutorial handbook.

Reeves, C. and J. Rowe (2003). *Genetic Algorithms: Principles and Perspectives*. Kluwer.

Reeves, C. and C. Wright (1994). An experimental design perspective on genetic algorithms. See Whitley (1994), pp. 7–22.

Reeves, C. R. (1999). Landscapes, operators and heuristic search. *Annals of Operational Research 86*, 473–490.

Reidys, C. M. and P. F. Stadler (1998). Neutrality in fitness landscapes. *Applied Mathematics and Computation 117*(2–3), 321–350.

Reshef, E. (1999, April). Approximating minimum communication cost spanning trees and related problems. Master's thesis, Feinberg Graduate School of the Weizmann Institute of Science, Rehovot 76100, Israel.

Ronald, S. (1995). *Genetic algorithms and permutation-encoded problems: Diversity preservation and a study of multimodality*. Unpublished doctoral dissertation, The University of South Australia.

Ronald, S. (1997). Robust encodings in genetic algorithms: A survey of encoding issues. In *Proceedings of the Forth International Conference on Evolutionary Computation*, Piscataway, NJ, pp. 43–48. IEEE.

Ronald, S., J. Asenstorfer, and M. Vincent (1995). Representational redundancy in evolutionary algorithms. In *1995 IEEE International Conference on Evolutionary Computation*, Volume 2, Piscataway, NJ, pp. 631–636. IEEE Service Center.

Rosenberg, R. S. (1967). *Simulation of genetic populations with biochemical properties*. Ph. D. thesis, The University of Michigan. (University Microfilms No. 67-17,836).

Rothlauf, F., J. Gerstacker, and A. Heinzl (2003). On the optimal communication spanning tree problem. Technical Report 15/2003, Department of Information Systems, University of Mannheim.

Rothlauf, F. and D. E. Goldberg (1999). Tree network design with genetic algorithms - an investigation in the locality of the prüfernumber encoding. In S. Brave and A. S. Wu (Eds.), *Late Breaking Papers at the Genetic and Evolutionary Computation Conference 1999*, Orlando, Florida, USA, pp. 238–244. Omni Press.

Rothlauf, F. and D. E. Goldberg (2000). Prüfernumbers and genetic algorithms: A lesson on how the low locality of an encoding can harm the performance of GAs. See Schoenauer, Deb, Rudolph, Yao, Lutton, Merelo, and Schwefel (2000), pp. 395–404.

Rothlauf, F., D. E. Goldberg, and A. Heinzl (2000). Bad codings and the utility of well-designed genetic algorithms. See Whitley, Goldberg, Cantú-Paz, Spector, Parmee, and Beyer (2000), pp. 355–362.

Rothlauf, F., D. E. Goldberg, and A. Heinzl (2001). On the debate concerning evolutionary search using Prüfer numbers. In A. S. Wu (Ed.), *Proceedings of the 2001 Genetic and Evolutionary Computaton Conference Workshop Program*, San Francisco, California, USA, pp. 262–267. Morgan Kaufmann.

Rothlauf, F., D. E. Goldberg, and A. Heinzl (2002). Network random keys – A tree network representation scheme for genetic and evolutionary algorithms. *Evolutionary Computation 10*(1), 75–97.

Rothlauf, F. and C. Tzschoppe (2004). Making the edge-set encoding fly by controlling the bias of its crossover operator. In G. Raidl and J. Gottlieb (Eds.), *Evolutionary Computation in Combinatorial Optimization 2005*, Volume 3348 of *LNCS*, pp. 202–212. Springer.

Rudnick, W. M. (1992). *Genetic algorithms and fitness variance with an application to the automated design of artificial neural networks*. Unpublished

doctoral dissertation, Oregon Graduate Institute of Science & Technology, Beaverton, OR.

Sastry, K. and Goldberg (2001). Modeling tournament with replacement using apparent added noise. IlliGAL Report No. 2001014, University of Illinois at Urbana-Champaign, Urbana, IL.

Schaffer, J. D. (Ed.) (1989). *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA. Morgan Kaufmann.

Schaffer, J. D., R. A. Caruana, L. J. Eshelman, and R. Das (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. See Schaffer (1989), pp. 51–60.

Schnier, T. and X. Yao (2000, 6-9 July). Using multiple representations in evolutionary algorithms. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, La Jolla Marriott Hotel La Jolla, California, USA, pp. 479–486. IEEE Press.

Schoenauer, M., K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel (Eds.) (2000). *Parallel Problem Solving from Nature, PPSN VI*, Berlin. Springer-Verlag.

Schuster, P. (1997). Genotypes with phenotypes: Adventures in an RNA toy world. *Biophys. Chem. 66*, 75–110.

Schwefel, H.-P. (1975). *Evolutionsstrategie und numerische Optimierung*. Ph. D. thesis, Technical University of Berlin.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. Chichester: John Wiley & Sons.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*. New York: Wiley & Sons.

Sendhoff, B., M. Kreutz, and W. von Seelen (1997a). Causality and the analysis of local search in evolutionary algorithms. Technical report, Institut für Neuroinformatik, Ruhr-Universität Bochum.

Sendhoff, B., M. Kreutz, and W. von Seelen (1997b). A condition for the genotype-phenotype mapping: Causality. In T. Bäck (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms*, San Francisco, pp. 73–80. Morgan Kaufmann.

Shackleton, M., R. Shipman, and M. Ebner (2000, 6-9 July). An investigation of redundant genotype-phenotype mappings and their role in evolutionary search. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, La Jolla Marriott Hotel La Jolla, California, USA, pp. 493–500. IEEE Press.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell Syst. Technical Jrnl. 27*, 379–423, 623–656.

Shannon, C. E. and W. Weaver (1949). *The Mathematical Theory of Communication*. Urbana, Illinois: University of Illinois Press.

Shipman, R. (1999). Genetic redundancy: Desirable or problematic for evolutionary adaptation? In *Proceedings of the 4th International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA)*, pp. 1–11. Springer Verlag.

Shipman, R., M. Shackleton, M. Ebner, and R. Watson (2000). Neutral search spaces for artificial evolution: A lesson from life. In M. Bedau, J. McCaskill, N. Packard, and S. Rasmussen (Eds.), *Proceedings of Artificial Life VII*, pp. section III (Evolutionary and Adaptive Dynamics). MIT Press.

Shipman, R., M. Shackleton, and L. Harvey (2000). The use of neutral genotype-phenotype mappings for improved evoutionary search. *British Telecom Technology Journal 18*(4), 103–111.

Sinclair, M. C. (1995). Minimum cost topology optimisation of the COST 239 European optical network. In D. W. Pearson, N. C. Steele, and R. F. Albrecht (Eds.), *Proceedings of the 1995 International Conference on Artificial Neural Nets and Genetic Algorithms*, New York, pp. 26–29. Springer-Verlag.

Smith, T., P. Husbands, and M. O'Shea (2001a). Evolvability, neutrality and search space. Technical Report 535, School of Cognitive and Computing Sciences, University of Sussex.

Smith, T., P. Husbands, and M. O'Shea (2001b). Neutral networks and evolvability with complex genotype-phenotype mapping. In *Proceedings of the European Converence on Artificial Life: ECAL2001*, Volume LNAI 2159, Berlin, pp. 272–281. Springer.

Smith, T., P. Husbands, and M. O'Shea (2001). Neutral networks in an evolutionary robotics search space. In I. of Electrical and E. Engineers (Eds.), *Proceedings of 2001 IEEE International Conference on Evolutionary Computation*, Piscataway, NJ, pp. 136–145. IEEE Service Center.

Spector, L., G. E., A. Wu, L. W. B., H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke (Eds.) (2001). *Proceedings of the Genetic and Evolutionary Computation Conference 2001*, San Francisco, CA. Morgan Kaufmann Publishers.

Stephens, C. R. and H. Waelbroeck (1999). Schemata evolution and building blocks. *Evolutionary Computation 7*, 109–124.

Streng, C. (1997, Juli). Optimierung eines bundesweiten Corporate Network am Beispiel der Firma DATEV. Master's thesis, Universität Erlangen-Nürnberg, Institut für Angewandte Mathematik.

Surry, D. and N. Radcliffe (1996). Formal algorithms + formal representations = search strategies. See Voigt, Ebeling, Rechenberg, and Schwefel (1996), pp. 366–375.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. See Schaffer (1989), pp. 2–9.

Tang, K. S., K. F. Man, and K. T. Ko (1997). Wireless LAN desing using hierarchical genetic algorithm. In T. Bäck (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms*, San Francisco, pp. 629–635. Morgan Kaufmann.

Thierens, D. (1995). *Analysis and design of genetic algorithms*. Leuven, Belgium: Katholieke Universiteit Leuven.

Thierens, D. and D. E. Goldberg (1993). Mixing in genetic algorithms. See Forrest (1993), pp. 38–45.

Thierens, D. and D. E. Goldberg (1994). Convergence models of genetic algorithm selection schemes. See Davidor, Schwefel, and Männer (1994), pp. 119–129.

Thierens, D., D. E. Goldberg, and Â. G. Pereira (1998). Domino convergence, drift, and the temporal-salience structure of problems. See Institute of Electrical and Electronics Engineers (1998), pp. 535–540.

Toussaint, M. and C. Igel (2002). Neutrality: A necessity for self-adaptation. In D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton (Eds.), *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pp. 1354–1359. IEEE Press.

Tzschoppe, C., F. Rothlauf, and H.-J. Pesch (2004). The edge-set encoding revisited: On the bias of a direct representation for trees. In Deb, Kalyanmoy et al. (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference 2004*, Heidelberg, pp. 1174–1185. Springer.

van Laarhoven, P. J. M. and E. H. L. Aarts (1988). *Simulated Annealing: Theory and Applications*. Dordrecht, The Netherlands: Kluwer.

Voigt, H.-M., W. Ebeling, I. Rechenberg, and H.-P. Schwefel (Eds.) (1996). *Parallel Problem Solving from Nature- PPSN IV*, Berlin. Springer-Verlag.

Vose, M. D. (1991). Generalizing the notion of a schema in genetic algorithms. *Artificial Intelligence 50*, 385–396.

Vose, M. D. (1993). Modeling simple genetic algorithms. See Whitley (1993), pp. 63–73.

Vose, M. D. (1999). *The simple genetic algorithm: foundations and theory*. Cambridge, MA: MIT Press.

Vose, M. D. and A. H. Wright (1998a). The simple genetic algorithm and the Walsh transform: Part I, theory. *Evolutionary Computation 6*(3), 253–273.

Vose, M. D. and A. H. Wright (1998b). The simple genetic algorithm and the Walsh transform: Part II, the inverse. *Evolutionary Computation 6*(3), 275–289.

Weinberger, E. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics 63*, 325–336.

Whitley, D. (1999). A free lunch proof for gray versus binary encodings. See Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela, and Smith (1999), pp. 726–733.

Whitley, D. (2000a). Functions as permutations: Implications for no free lunch, walsh analysis and statistics. See Schoenauer, Deb, Rudolph, Yao, Lutton, Merelo, and Schwefel (2000), pp. 169–178.

Whitley, D. (2000b). Local search and high precision gray codes: Convergence results and neighborhoods. See Martin and Spears (2000), pp. unknown. in press.

Whitley, D. (2002, September). Evaluating evolutionary algorithms. Tutorial Program at Parallel Problem Solving from Nature (PPSN 2002.

Whitley, D., D. E. Goldberg, E. Cantú-Paz, L. Spector, L. Parmee, and H.-G. Beyer (Eds.) (2000). *Proceedings of the Genetic and Evolutionary Computation Conference 2000*, San Francisco, CA. Morgan Kaufmann Publishers.

Whitley, D. and S. Rana (1997). Representation, search, and genetic algorithms. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pp. 497–502. AAAI Press/MIT Press.

Whitley, D., S. Rana, and R. Heckendorn (1997). Representation issues in neighborhood search and evolutionary algorithms. In *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, Chapter 3, pp. 39–58. West Sussex, England: John Wiley & Sons Ltd.

Whitley, L. D. (Ed.) (1993). *Foundations of Genetic Algorithms 2*, San Mateo, CA. Morgan Kaufmann.

Whitley, L. D. (Ed.) (1994). *Foundations of Genetic Algorithms 3*, San Francisco. Morgan Kaufmann Publishers, Inc.

Wolpert, D. H. and W. G. Macready (1995). No free lunch theorems for search. Tech. Rep. No. SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM.

Wong, R. (1980). Worst case analysis of network design problem heuristics. *SIAM J. Algebraic Discr. Meth. 1*, 51–63.

Wu, B. Y., G. Lancia, Y. Bafna, K. M. Chao, R. Ravi, and C. Y. Tang (1998, January). A polynomial time approximation schem for minimum routing cost spanning trees. In *Proc. 9th ACM-SIAM Symp. on Discrete Algorithms*, pp. 21–32.

Yu, T. and J. Miller (2001). Neutrality and evolvability of Boolean function landscapes. In *Proceedings of the 4th European Conference on Genetic Programming (EuroGP)*, Volume LNCS 2038, pp. 204–217. Springer.

Yu, T. and J. Miller (2002). Finding needles in haystacks is not hard with neutrality. In *Proceedings of the 5th European Conference on Genetic Programming (EuroGP)*, Volume LNCS, pp. 13–25. Springer.

Zhou, G. and M. Gen (1997). Approach to degree-constrained minimum spanning tree problem using genetic algorithm. *Engineering Design & Automation 3*(2), 157–165.

List of Symbols

α	probability of GEA failure
$\bar{\alpha}$	average percentage of incorrect alleles
α_i	ith coefficients of the polynomial decomposition of $\boldsymbol{x}$
$\boldsymbol{b}$	biased chromosome
b_{ij}	amount of traffic running over link from node i to node j
c	crossover-point in string
$c(T)$	communication cost of a spanning tree T
$\mathbf{c}$	characteristic vector
cap_{ij}	capacity of a link between node i and j
d	signal difference / distance between nodes / distance between individuals
d_c	distance distortion of an encoding
d^g	genotypic distance
d^h	Hamming distance
d_i	length of a link i
d_m	locality of an encoding
d_{MST}	distance of an individual towards the MST
d^p	phenotypic distance
d_{ij}	distance weight (cost) of a link (ij)
$d_{\boldsymbol{x},\boldsymbol{y}}$	distance between individual $\boldsymbol{x}$ and $\boldsymbol{y}$
$\delta(h)$	defining length of a schema h
D	distance matrix
$deg(i)$	degree of node i
E	set of edges
e_{ij}	edge between node i and j
$E(x)$	mean of x
f_g	genotype-phenotype mapping (representation)

f_p	phenotype-fitness mapping
$f(\boldsymbol{x})$, f_x	fitness of individual $\boldsymbol{x}$
$f(h,t)$	fitness of schema h at time t
$\bar{f}(t)$	average fitness of population at time t'
G	graph
γ_{avg}	average number of genotypes
h	schema (ternary string of length l, where $h_i \in \{0,1,*\}$
h_i	ith allele of a schema h
I	selection intensity
k	size or oder of BB
k_d	number of parental edges
k_g	genotypic size of BB
k_p	phenotypic size of BB
k_r	order of redundancy
l	length of a string
l_g	length of a genotypic bitstring
l_p	length of a phenotypic bitstring
l_s	genotypic length of an exponentially scaled BB
$l_{i,j}$	link between node i and j
λ	dividing line between converged and unconverged alleles (domino convergence model) / order of schema (locality)
λ_c	size of the convergence window
m	number of BBs
m'	m-1
$m(h,t)$	number of instances of schema h at time t
μ	mean
n	number of nodes
n_f	number of fitness calls
n_p	number of different phenotypes
N	population size
N_{drift}	population size necessary for GAs not to be affected by genetic drift
$\mathbb{N}$	integer numbers
$\mathbb{N}(\mu,\sigma)$	normal distribution with mean μ and standard deviation σ
$o(h)$	order or size of a schema h
p	probability
p_c	probability of crossover
p_m	probability of mutation
P_1	link-specific bias
P_2	node-specific bias

$P_n,\ P_{succ}$	probability of GEA success
$p_{i,j}^T$	unique path from node i to node j in tree T
$\psi_j(\boldsymbol{x})$	jth Walsh function for $\boldsymbol{x}$
$\boldsymbol{\Phi}_g$	genotypic search space
$\boldsymbol{\Phi}_p$	phenotypic search space
q	probability of making the wrong decision when deciding between a single sample of each BB
r	number of genotypic BBs that represent the optimal phenotypic BB / order of a binary encoded problem
r	random key vector
r^s	permutation vector
r_{ij}	communication requirement
$\mathbb{R}$	real numbers
s	order of scaling / size of tournament / number of possibilities
$s(t)$	probability that an allele is fully converged
σ	standard deviation
σ_{BB}	standard deviation of a BB
σ_f	standard deviation of function f
σ_N	standard deviation of additional noise
t	time, number of generations
t_{conv}	convergence time
t_{drift}	drift time
T	spanning tree / temperature (for simulated annealing)
u	number of ones
$u(\boldsymbol{x})$	number of ones in $\boldsymbol{x}$
V	set of nodes
w	Walsh coefficients
χ	cardinality of alphabet
x_0	expected number of copies of the best BB in a randomly initialized population
x_i	ith allele of an individual $\boldsymbol{x}$
$\boldsymbol{x}$	vector of decision variables / individual
x^g	genotype of an individual
x^p	phenotype of an individual
x_i^c	contribution of the ith most salient allele to the fitness of the individual
*	don't care symbol

List of Acronyms

BB	building block
CV	characteristic vector
GA	genetic algorithm (meaning selectorecombinative GEAs)
GEA	genetic and evolutionary algorithm (meaning GEAs using crossover and mutation)
ES	evolution strategy
eTV	extended trivial voting
LB	link biased
LNB	link and node biased
MST	minimum spanning tree
NB	node biased
NetDir	direct tree
NetKey	network random key
OCST	optimal communication spanning tree
RK	random key
SA	simulated annealing
TV	trivial voting

Index