Memetic Algorithms for Combinatorial Optimization Problems:

Fitness Landscapes and Effective Search Strategies

Vom Fachbereich Elektrotechnik und Informatik
der Universität-Gesamthochschule Siegen
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

genehmigte Dissertation

von

Diplom-Ingenieur Peter Merz
aus Kreuztal

1. Gutachter: Prof. Dr.-Ing. Bernhard Freisleben
2. Gutachter: Prof. Dr. rer. nat. Wolfgang Merzenich

Tag der mündlichen Prüfung: 21. Dezember 2000

Acknowledgements

The material presented in this thesis is based on the research I have done during my time as a research assistant in the Parallel Systems Research Group at the Department of Electrical Engineering and Computer Science at the University of Siegen.

I would like to thank all those collaborators who made this research not only possible, but also worthwhile and enjoyable. The following persons I thank individually.

First of all, I have to thank my academic mentor Bernd Freisleben who introduced me to the secrets of academic work and who gave me the freedom to focus my research in the direction I asked for. His financial support for attending several international conferences allowed me to get in touch with other researchers in the field of evolutionary computation and combinatorial optimization. I have learned so much by discussing with my international colleagues – not only about our common research subjects.

Many thanks go to Wolfgang Merzenich, my second referee, who showed much interest in my work. Furthermore, I wish to thank Udo Kelter for his part-time financial support in the first year of my research.

Several other people at our department have supported my work through all these years. I'd like to thank Frank Thilo who provided his Winner System as well as the necessary support to get things running. Moreover, I'd like to thank André Treptow, Patrick Euteneuer and Jörg Kühn for their extensions and refinements to COPS, the software package containing the algorithms I developed during my work.

Several researchers around the globe also contributed to the production of my thesis. First of all, I'd like to thank Martina Gorges-Schleuter for her excellent work which inspired me and shifted my focus to hybrid evolutionary algorithms. Discussions with her were always fruitful. Furthermore, I'd like to thank Pablo Moscato for his many comments and suggestions and the opportunity to contribute a chapter to the book *New Ideas in Optimization*. Thomas Stützle provided lots of suggestions and helpful criticism as well as the source code of his algorithms. Thanks to him and to Kengo Katayama for his ideas and support; especially for his refinements to the k-opt local search for the BQP. Many thanks also go to Tim Walters for the fruitful discussions on evolutionary algorithms for the TSP, to Andy Reinholz for the discussions and material support for vehicle routing, and to Volker Schnecke for 'giving me shelter' during the ICGA'97 conference. These people always believed in me and provided a constant source of encouragement. Without them and without the support of my family I would not have come this far.

Finally, I'd like to thank all my colleagues at the department of computer science in Siegen. They provided a very friendly atmosphere and ensured that working at the department was always fun.

Zusammenfassung

Kombinatorische Optimierungsprobleme sind in vielen Bereichen der Forschung und Entwicklung zu finden. Betriebswirtschaftlich ausgedrückt, handelt es sich um Entscheidungssituationen, in denen ein vorgegebener Nutzen in kostenminimaler Weise zu realisieren ist, wobei aus mehreren Alternativen eine nach speziellen Kriterien optimale ausgewählt wird. Voraussetzung ist, daß sich das Problem auf ein mathematisches Modell abbilden läßt, das jeder Alternative – einer Lösung des Problems – eine Güte zuordnet. Die Ermittlung des Fertigungssplans mit der höchsten Produktivität, die Entwicklung eines Telekommunikationsnetzwerkes mit der höchst möglichen Zuverlässigkeit, oder die Vorhersage der räumlichen Struktur eines Proteins nach seiner Faltung sind nur einige Beispiele kombinatorischer Optimierungsprobleme. Viele ähnliche Probleme sind aus den Bereichen der Betriebswirtschaft, dem Maschinenbau, der Elektrotechnik, der Informatik, der Biologie und der Physik bekannt.

Im Gegensatz zu anderen Optimierungsproblemen haben kombinatorische Optimierungsprobleme eine endliche Zahl möglicher Lösungen. Daher können prinzipiell optimale Lösungen durch das Betrachten aller möglichen Lösung gefunden werden. Dieses Vorgehen der vollständigen Enumeration erweist sich aber nur selten als praktikabel, da bei den meisten kombinatorischen Problemen die Anzahl der möglichen Lösungen einfach zu groß ist. Einige der in dieser Arbeit betrachteten Optimierungsprobleme haben mehr als 10^{9259} mögliche Lösungen, wogegen die geschätzte Anzahl der Atome in unserem Universum mit 10^{80} als unbedeutend klein erscheint. Für einige kombinatorische Optimierungsprobleme sind Algorithmen gefunden worden, die deutlich schneller als die vollständige Enumeration sind. Man sagt, daß sie eine polynomiale Laufzeit in Abhängigkeit der Problemgröße besitzen, falls die Laufzeit durch ein Polynom in n (der Problemgröße) ausgedrückt werden kann. Für viele andere *harte* kombinatorische Optimierungsprobleme ist kein solch schneller Algorithmus bekannt. Es wird gemeinhin angenommen, daß kein Algorithmus mit polynomialer Laufzeit für diese Probleme existiert, auch wenn ein Beweis für diese Annahme aussteht. Dennoch hat man in den vergangenen Jahren enorme Fortschritte in der exakten Lösung dieser Probleme mittels hoch entwickelter Verfahren erzielt. Doch die Komplexität der Probleme bleibt. Daher ist man in den meisten Fällen darauf angewiesen, Heuristiken einzusetzen, die zwar die Ermittlung einer besten (optimalen) Lösung nicht garantieren, aber in der Praxis mit hoher Wahrscheinlichkeit das Optimum oder ein Suboptimum mit geringfügig schlechterer Lösungsgüte finden. Ein Teilbereich der Informatik hat sich zum Ziel gesetzt, neue Heuristiken für kombinatorische Optimierungsprobleme zu finden, bereits bekannte zu verbessern, und verschiedene Verfahren zu kombinieren.

Heuristiken für kombinatorische Optimierungsprobleme können in problemspezifische und problemunabhängige Heuristiken eingeteilt werden. Beispiele für problemunabhängige Techniken sind Nachbarschaftssuche, wie lokale Suche und Tabu-Suche, sowie biologisch inspirierte Methoden, wie evolutionäre Algorithmen, simulierte Ameisenkolonien, oder künstliche neuronale Netze. In den letzten Jahren hat man erkannt, daß die Kombination aus

problemspezifischen und problemunabhängigen Heuristiken besonders vielversprechend ist: Durch Einbringen von problemspezifischem Wissen in evolutionäre Algorithmen kann sich in vielen Fällen die Effektivität des Ansatzes erheblich verbessern. Dabei werden Synergieeffekte deutlich: Die Kombination zeigt eine deutliche Effektivitätssteigerung gegenüber den Einzelkomponenten.

Die Forschung im Bereich der heuristischen Lösung kombinatorischer Optimierungsprobleme konzentriert sich überwiegend auf die Entwicklung von neuen, effektiveren Varianten bestehender Heuristiken, welche häufig nur auf einigen, zumeist selbst generierten Instanzen eines ausgesuchten Optimierungsproblems getestet werden. Ferner werden diese Heuristiken mehr oder weniger adäquat mit anderen Ansätzen verglichen, um ihre Überlegenheit zu "beweisen". Das Ziel der vorliegenden Arbeit ist nicht, an der Suche nach der "besten" Heuristik zu partizipieren – es ist anzunehmen, daß keine solche existiert. Stattdessen beschäftigt sich die Dissertation mit den folgenden Fragen: Warum ist eine gegebene Heuristik auf einer gegebenen Menge von Testinstanzen effektiver als eine andere? Was sind die entscheidenden Eigenschaften von Probleminstanzen, die sich auf die Effektivität von Heuristiken auswirken? Wie kann man problemspezifisches Wissen einsetzen, um eine Heuristik für ein Problem effektiver und effizienter zu machen? Läßt sich das zu lösende Problem in Instanzklassen einteilen, und kann man das vorhandene Problemwissen dazu einsetzen, adäquatere Testfälle für Heuristiken zu erzeugen?

Um Antworten auf die obigen Fragen zu finden, konzentriert sich diese Arbeit auf eine bestimmte Klasse von Heuristiken: die der memetischen Algorithmen. Es gibt mehrere Gründe für diese Wahl. Zum einen erlauben memetische Algorithmen das Einsetzen von problemspezifischem Wissen in einer kontrollierten Art und Weise, ohne daß dabei die zu Grunde liegende Methodik verloren geht. Weiterhin profitieren memetische Algorithmen von den Vorteilen eines hybriden Ansatzes. Sie nutzen die symbiotischen Effekte der unterschiedlichen Suchstrategien, die sie verwirklichen: Sie beinhalten Nachbarschaftssuche, die sich sehr gut für die intensive Suche in einem ausgewählten Gebiet eignet, sowie die populationsbasierte evolutionäre Variation, wie sie in evolutionären Algorithmen zu finden ist, die der Identifikation vielversprechender Gebiete im Suchraum dient. Ein weiterer Grund für die getroffene Wahl ist, daß sich memetische Algorithmen als sehr effektive Algorithmen erwiesen haben. Sie sind unter den effektivsten bis heute entwickelten Heuristiken für eine Reihe von kombinatorischen Optimierungsproblemen.

Der Versuch, Einblicke in die Struktur von kombinatorischen Optimierungsproblemen und in die Dynamik von heuristischen Suchprozessen zu erlangen, ist der erste Schritt, um Antworten auf die oben gestellten Fragen zu finden. Das aus der Evolutionstheorie entliehene Konzept der Fitness-Landschaften ist in diesem Zusammenhang äußerst hilfreich. Die Idee ist dabei, die Menge aller Lösungen – den Suchraum – als eine räumliche Struktur aufzufassen. Jeder Punkt in dieser räumlichen Struktur repräsentiert eine Lösung. Die Fitness (Lösungsgüte) wird durch die Höhe des Punktes, der sie repräsentiert, dargestellt. Somit ergibt sich ein Gebirge bzw. eine Landschaft, wenn man im Lösungsraum benachbarte Punkte miteinander verbindet. Die Gebirgsspitzen stellen folglich lokale Optima und die höchste Gebirgsspitze das globale Optimum dar. Einige Eigenschaften dieser Fitness-Landschaften beeinflussen stark die Effektivität und die Effizienz der Nachbarschaftssuche sowie der evolutionären (Meta-)Suche eines memetischen Algorithmus. Für die erstere ist die lokale Struktur von entscheidender Bedeutung, die mit einer statistischen Autokorrelationsanalyse untersucht werden kann. Die globale Struktur, die sich unter anderem in der Verteilung der lokalen Optima, also der Gebirgsspitzen in der Fitness-Landschaft, ausdrückt, hat auf die evolutionäre Komponente des memetischen Algorithmus entscheidenden Einfluß. Zur Analyse der globalen Struktur ist die statistische Fitness-Distanz-Korrelationsanalyse ein hilfreiches Werkzeug.

Um die Verbindung zwischen Fitness-Landschaften von Optimierungsproblemen und der Effektivität von memetischen Algorithmen systematisch zu untersuchen, werden fünf verschiedene Probleme in dieser Arbeit betrachtet: *NK*-Fitness-Landschaften (*NK-Landscapes*), das unbeschränkte binäre quadratische Optimierungsproblem (*BQP*), das Problem des Handlungsreisenden (*TSP*), das Graphen-Bipartitionierungsproblem (*GBP*) und das quadratische Zuweisungsproblem (*QAP*).

Die Arbeit ist wie folgt gegliedert. Nach einer kurzen Einführung wird in Kapitel 2 ein Überblick über kombinatorische Optimierungsprobleme gegeben. Es werden die bekanntesten Vertreter beschrieben, exakte Verfahren und Heuristiken diskutiert und eine Klassifikation bezüglich der beinhalteten Nebenbedingungen getroffen, welche bei der Entwicklung von Heuristiken eine große Rolle spielt. Im dritten Kapitel werden moderne heuristische Techniken beschrieben. Hier liegt der Schwerpunkt auf evolutionären Algorithmen und der Nachbarschaftssuche, sowie der Kombination aus beiden, den memetischen Algorithmen. Eigenschaften von Fitness-Landschaften und deren statistische Analyse sind Gegenstand von Kapitel 4. Der Fokus liegt auf der Autokorrelationsanalyse und der Fitness-Distanz-Korrelationsanalyse. Es wird diskutiert, wie diese Techniken genutzt werden können, um die Effektivität eines Ansatzes vorherzusagen, bzw. wie die Ergebnisse für die Entwicklung effektiver memetischer Algorithmen verwertet werden können. In den Kapiteln 5 bis 9 werden nacheinander *NK*-Fitness-Landschaften, das unbeschränkte binäre quadratische Optimierungsproblem, das Problem des Handlungsreisenden, das Graphen-Bipartitionierungsproblem und das quadratische Zuweisungsproblem detailliert untersucht. In jedem dieser Kapitel werden Heuristiken für die Probleme beschrieben mit Schwerpunkt auf *Greedy*-Heuristiken und Nachbarschaftssuche, da diese gut für die Verwendung in einem memetischen Algorithms geeignet sind. Zusätzlich wird eine Analyse mit den in Kapitel 4 eingeführten statistischen Werkzeugen durchgeführt. Hier kommen die Autokorrelationsanalyse und die Fitness-Distanz-Korrelationsanalyse zum Einsatz. Daran anschließend werden unter Berücksichtigung der Analyseergebnisse memetische Algorithmen entwickelt, deren Effektivität in Computersimulationen gezeigt wird. Im abschließenden Kapitel wird die Arbeit zusammengefaßt und es werden kapitelübergreifende Schlüsse gezogen. Das Kapitel und die Dissertation enden mit einem Ausblick auf zukünftige Forschungsrichtungen.

In der vorliegenden Arbeit werden systematisch die fünf betrachteten kombinatorischen Optimierungsprobleme analysiert. Für drei der fünf Probleme ist die Autokorrelation mathematisch von anderen Forschern ermittelt worden. Für das binäre quadratische Optimierungsproblem und für das quadratische Zuweisungsproblem wird in dieser Arbeit die Autokorrelationsanalyse experimentell durchgeführt. Zusätzlich wird für alle fünf Probleme die Fitness-Distanz-Korrelation experimentell bestimmt. Weiterhin werden für alle Probleme neue evolutionäre Variationsoperatoren und/oder Greedy- und lokale Suchheuristiken vorgeschlagen: Für *NK*-Landschaften werden eine neue Greedy-Heuristik, eine k-optimale lokale Suche und ein heuristischer Rekombinationsoperator eingeführt. Eine neue Greedy-Heuristik und eine k-optimale lokale Suche werden ebenfalls für das binäre quadratische Optimierungsproblem entwickelt. Für das Problem des Handlungsreisenden werden zwei neue Rekombinationsoperatoren definiert, ein distanzerhaltender Rekombinationsoperator und ein generischer, heuristischer Rekombinationsoperator. Im Falle des Graphen-Bipartitionierungsproblems wird ein neuer heuristischer Rekombinationsoperator in Anlehnung an eine Greedy-Heuristik vorgeschlagen. Für das quadratische Zuweisungsproblem werden ebenfalls zwei Rekombinationsoperatoren eingeführt, ein distanzerhaltender Rekombinationsoperator und eine zyklischer Rekombinationsoperator in Anlehnung an den *Cycle-Operator* für das Problem des Handlungsreisenden. In allen fünf Fällen wird gezeigt, daß memetische Algorithmen, die diese neuen Operatoren und Heuristiken einsetzen, zu den besten bisher veröffentlichten Heuristiken für das jeweilige Problem zählen.

Bezüglich der oben gestellten Fragen lassen sich sowohl aus den Analyseexperimenten als auch aus den Experimenten mit den memetischen Algorithmen Schlüsse ziehen. Zusammengefaßt ergeben sich aus der Analyse die folgenden Aussagen.

- Die Eigenschaften der Fitness-Landschaften sind keinesfalls für alle Probleme gleich. Sie variieren stark von Problem zu Problem, und es gibt innerhalb eines Problems sehr stark unterschiedliche Instanzen, die sogar komplementäre Eigenschaften besitzen können.

- Instanzen mit regelmäßiger Struktur oder Instanzen deren Optima in polynomialer Zeit berechnet werden können, haben nicht die charakteristischen Eigenschaften von in Anwendungen auftretenden Instanzen. In den betrachteten Fällen haben sie sich (für Heuristiken) als deutlich leichter lösbar erwiesen.

- Die Effektivität von Heuristiken bei zufällig generierten Instanzen unterscheident sich deutlich von der Effektivität bei strukurierten Instanzen, die in Anwendungen auftreten. Insbesondere haben Instanzen, bei denen die Daten eine Gleichverteilung annehmen, völlig andere Eigenschaften. Ein Algorithmus, der auf diesen Instanzen gute Ergebnisse liefert, muss dies nicht notwendigerweise auch auf anderen Instanzen tun.

- Daraus ergibt sich die Forderung, daß Testinstanzen Vertreter aller Typen von Fitness-Landschaften eines Problems beinhalten sollten.

- Der Größe der Region des Suchraums, in der die lokalen Optima liegen, kommt eine große Bedeutung zu. Sie variiert stark von Problem zu Problem. Effektive Heuristiken sollten dies berücksichtigen.

- Hohe Epistasis – ein Maß für die Interaktion zwischen den Komponenten eines Lösungsvektors und somit ein Maß für die Nichtlinearität des Problems – bedeutet nicht immer, wie bisher angenommen, daß ein Problem (für Heuristiken) schwer lösbar ist.

Die Schlußfolgerungen aus den Experimenten mit den memetischen Algorithmen lassen sich ebenfalls zusammenfassen:

- Memetische Algorithmen skalieren viel besser mit der Problemgröße als evolutionäre Algorithmen oder wiederholte lokale Suche. Insbesondere lassen sich evolutionäre Algorithmen ohne heuristische Elemente nur auf Probleme mit einer Problemgröße, die in der Praxis uninteressant ist, effektiv anwenden.

- *Greedy*-Komponenten sind für Probleme mit geringer Epistasis in einem memetischen Algorithmus vielversprechend. Sie können bei der Initialisierung der Population und bei der Rekombination verwendet werden.

- Memetische Algorithmen erweisen sich am effektivsten, wenn die Fitness-Landschaft einer Instanz Korrelation aufweist, d.h. wenn eine globale Struktur erkennbar ist. In diesem Fall kann Rekombination als Mittel der evolutionären Variation gewinnbringend eingesetzt werden, da sie die Struktur ausnutzt und die Suche auf den Bereich des Suchraums mit überdurchschnittlichen Lösungen konzentriert.

- Wenn die Fitness-Landschaft korreliert ist, die lokalen Optima im Suchraum aber zu dicht beieinander liegen, wird evolutionäre Suche mittels Rekombination ineffektiv, da die Diversifikationseigenschaft zu schnell mit der Konvergenz der Suche abnimmt.

- Es hat sich gezeigt, daß iterierte lokale Suche beim Problem des Handlungsreisenden sehr effektiv ist. In den anderen untersuchten Problemen ist sie allerdings der populationbasierten Suche unterlegen.

Wie die vorliegende Dissertation zeigt, ist es möglich, die Effektivität von Rekombination im Vergleich zu Mutation als Mittel der Variation in memetischen Algorithmen in gewissem Maße vorherzusagen, wenn eine Analyse der Fitness-Landschaft als Grundlage dient.

Teile der Ergebnisse zu den *NK*-Fitness-Landschaften und der binären quadratischen Optimierung in den Kapiteln 5 und 6 sind in den Konferenzbänden der *IEEE International Conference on Evolutionary Computation* (1998) [212] und in den Tagungsbänden der *International Genetic and Evolutionary Computation Conference (GECCO)* (1999) [215] erschienen. Einige der Heuristiken für das binäre quardratische Optimierungsproblem sind in einem Arikel beschrieben, der in der Fachzeitschrift *Journal of Heuristics* erscheint [217]. Frühere Ergebnisse des memetischen Algorithmus für das Problem des Handlungsreisenden, wie in Kapitel 7 beschrieben, sind in den Konferenzbänden der *IEEE Conference on Evolutionary Computation (ICEC)* (1996 und 1997) [105, 210] und in den Tagungsbänden der *Fourth International Conference on Parallel Problem Solving from Nature* (1996) [106] veröffentlicht. Teilergebnisse der Untersuchungen für das Graph-Bipartitionierungsproblem in Kapitel 8 sind in den Tagungsbänden der *Fifth International Conference on Parallel Problem Solving from Nature (PPSN)* (1998) [211] erschienen. Ein ausführlicher Artikel ist in der Zeitschrift *Journal of Evolutionary Computation* [217] publiziert. Erste Ergebnisse für das quadratische Zuweisungsproblem sind in den Konferenzbänden *Seventh International Conference on Genetic Algorithms and their Applications (ICGA)* (1997) [209] und *Congress on Evolutionary Computation (CEC)* (1999) [213] zu finden. Ein ausführlicher Artikel zu den Resultaten in Kapitel 9 ist in der Fachzeitschrift *IEEE Transactions on Evolutionary Computation* veröffentlicht [216]. Schließlich ist zur Fitness-Landschaft-Analyse und der Entwicklung effektiver memetischer Algorithmen ein Kapitel in dem Buch *New Ideas in Optimization* [214] erschienen. Der Artikel enthält Grundlagen aus Kapitel 4 sowie frühe Teilergebnisse zum Problem des Handlungsreisenden, Graph-Bipartitionierungsproblem und zum quadratischen Zuweisungsproblem.

Contents

Chapter 1

Introduction

Combinatorial optimization problems are found in many areas of research and development. They arise when the task is to find the best out of many possible solutions to a given problem, provided that a clear notion of solution quality exists. Finding the factory production schedule with the highest throughput, designing the most reliable telecommunications network, or finding the structure of a protein molecule in the three dimensional space that minimizes potential energy are just few examples of combinatorial optimization problems. Many others have been reported in the fields of management science, industrial engineering, computer science, biology, and physics.

In contrast to other optimization problems, combinatorial problems have a finite number of candidate solutions. Therefore, an obvious way to solve these problems is to enumerate all candidate solutions by comparing them against each other. Unfortunately, for most interesting combinatorial optimization problems, this approach proves to be impractical since the number of candidate solutions is simply too large. For example, some of the problems considered in this thesis have more than 10^{9259} candidate solutions which is an extraordinary high number compared to the estimated number of 10^{80} atoms in the universe. For some combinatorial problems, algorithms have been found which are much faster than exhaustive search: they are said to run in polynomial time depending on the problem size. For many other *hard* combinatorial optimization problems it is commonly believed that there is no such fast algorithm. However, in recent years enormous progress has been made in solving these problems with exact algorithms. But the computational complexity still remains: for the vast majority of cases, the only way to tackle the problems is to apply heuristic search that delivers no guarantee of finding the optimum solution. Consequently, an enormous effort has been made in developing heuristics that are aimed at finding high quality solutions in short time.

Heuristics for combinatorial optimization problems can be separated into problem-specific algorithms and problem-independent methods. Examples of modern problem-independent techniques are neighborhood search algorithms such as local search, tabu search, or simulated annealing, and biologically inspired methods like evolutionary algorithms, ant colony systems, and neural nets. Even hybrid methods exist that combine two or more search strategies, e.g. memetic algorithms which are hybrids of neighborhood search methods and evolutionary algorithms. In recent years, researchers focused on the development of new, more effective variants of heuristics which are tested on some, often self-generated instances of a selected problem. They are compared more or less adequately with alternative approaches to ‘prove’ superior effectiveness.

The goal of this thesis is not to participate in the quest of finding the ‘best heuristic ever developed' - in fact it is commonly believed that there is no such 'best' heuristic that is superior to all other heuristics on all problems. Instead, the research is focused on the following important questions: why does a given heuristic perform better on a given set of problem instances than others? What are the key problem characteristics that make it hard for a certain class of heuristics? Is there a way to predict the performance of a heuristic on a particular problem? How can we use knowledge about the problem to design more effective algorithms? Can we employ knowledge of the problem to determine better test cases?

In an attempt to find answers to the questions, this thesis is focused on a particular class of heuristics: memetic algorithms. There are several reasons for this choice. Many researchers experienced that it is very important to incorporate domain-specific knowledge into problem-independent algorithms. Memetic algorithms allow to do this in a controlled manner, keeping the basic ideas behind the memetic approach. Moreover, memetic algorithms are hybrids. They exploit the symbiotic effects of the combination of two (sometimes more) different search strategies: they incorporate neighborhood search algorithms that are well-suited for intensifying search while the evolutionary framework enables effective diversification. Finally, memetic algorithms have been shown to be among the most effective heuristics for combinatorial optimization problems to date.

Gaining insight into the structure of combinatorial problems is the first step in finding answers to the questions stated above. The concept of fitness landscapes borrowed from biologists has proven to be very useful in the context of optimization. The basic idea is to view the set of all candidate solutions – the search space – as a spatial structure in which each point represents a candidate solution. Each point has a height that reflects the quality (fitness) of the represented solution. The spatial arrangement based on a well-defined neighborhood structure yields a fitness landscape since the heights vary from one point to the other. Some characteristics of fitness landscapes are strongly related to the performance of the neighborhood search as well as to the evolutionary meta-search of the memetic algorithm. For the former, the local structure of the fitness landscape is of great importance which can be investigated with an autocorrelation or random-walk correlation analysis. The effectiveness of the latter is highly influenced by the distribution of the points in the search space which are produced by the neighborhood search. A fitness distance correlation analysis can be utilized to find characteristics in this distribution.

To investigate the relation between the fitness landscapes of combinatorial optimization problems and the performance of memetic algorithms, five problems are studied in detail, each of which covers different aspects of problem difficulty: NK-landscapes, unconstrained binary quadratic programs, the traveling salesman problem, the graph bipartitioning problem, and the quadratic assignment problem. Therefore, a more general view of the landscape characteristics and algorithm performance is provided than by other studies in the field. For each of the problems, heuristics that can be incorporated into memetic algorithms are discussed, a fitness landscape analysis is performed to reveal important problem characteristics, and the effectiveness of memetic algorithms is investigated in experiments.

The thesis is organized as follows. In chapter 2, a brief overview of combinatorial optimization problems is provided: well-known examples of these problems are described, exact methods and heuristics are discussed, and a classification of combinatorial problems is given. Chapter 3 describes modern heuristic techniques with emphasis on evolutionary algorithms and neighborhood search as well as their combination: memetic algorithms. Techniques for analyzing fitness landscapes are described in chapter 4. The focus is put on autocorrelation/random-walk analysis and fitness distance correlation analysis. It is discussed how these techniques can be utilized for the performance prediction or the design of memetic algorithms. In the chapters 5 through 9, *NK*-landscapes, unconstrained binary quadratic programs, the traveling salesman problem, the graph bipartitioning problem, and the quadratic assignment problem are investigated, respectively. In each of these chapters, heuristics for the problems are described with emphasis on greedy and local search heuristics, since these two types of heuristics are well-suited for the incorporation into an evolutionary framework. Additionally, a search space analysis of the problems is performed by employing the techniques discussed in chapter 4. The performance of memetic algorithms for the particular problem is evaluated in computer experiments afterwards. In the final chapter, the results obtained from the experiments described in the various chapters are compared, and general conclusions are drawn. The thesis finishes with a discussion of important future work.

Parts of the results on *NK*-landscapes and binary quadratic programming in the chapters 5 and 6 have been published in the proceedings of the *1998 IEEE International Conference on Evolutionary Computation* [212], and in the proceedings of the *International Genetic and Evolutionary Computation Conference (GECCO)* [215], respectively. The heuristics developed for the binary quadratic programming problem are described in an article which is accepted for publication in the *Journal of Heuristics* [218]. Early results of the memetic algorithm for the traveling salesman problem described in chapter 7 have been published in the proceedings of the *IEEE Conference on Evolutionary Computation (ICEC)* in the years 1996 and 1997 [105, 210], and in the proceedings of the *Fourth International Conference on Parallel Problem Solving from Nature* [106]. Partial results of the studies on the graph bipartitioning problem presented in chapter 8 have been published in the proceedings of the *Fifth International Conference on Parallel Problem Solving from Nature (PPSN)* [211]. A full-length paper appears in the *Journal of Evolutionary Computation* [217]. Preliminary results on the quadratic assignment problem have been published in the proceedings of *Seventh International Conference on Genetic Algorithms and their Applications (ICGA)* [209] and in the proceedings of the *1999 Congress on Evolutionary Computation (CEC)* [213]. A self-contained version of chapter 9 has been published in the journal *IEEE Transactions on Evolutionary Computation* [216]. Finally, a chapter on the fitness landscape analysis and the design of memetic algorithms containing research results of an earlier stage of this work have been published in the book *New Ideas in Optimization* [214].

Chapter 2

Combinatorial Optimization Problems

2.1 Introduction

Combinatorial optimization problems (COPs) arise in many practical applications in the fields of management science, biology, chemistry, physics, engineering, and computer science. Project and resource management, transportation management, capital budgeting, network routing, protein folding/molecular conformation, x-ray crystallography, spin glass models, and VLSI design and fabrication are just a few examples of fields in which combinatorial optimization problems occur.

Many of these problems are very complex and thus hard to solve; general mathematical methods are not available. Often, the number of candidate solutions of an optimization problem instance grows exponentially with the problem size so that simple enumeration schemes are rendered impractical. Thus, combinatorial optimization problems constitute a class of problems with high practical importance but extreme hardness with respect to the solution of these problems.

In this chapter, a formal definition of COPs is given to distinguish them from other optimization problems. Several examples of well–known combinatorial problems in operations research and their application areas are provided afterwards. Furthermore, exact methods for solving COPs are described, and the importance of heuristics for these problems is discussed. Finally, a classification of COPs based on the characteristics of the constraints present in a problem is provided.

2.2 Definitions

According to [113], a combinatorial optimization problem P is either a minimization problem or a maximization problem, and it consists of

(i) a set D_P of instances,

(ii) a finite set $S_P(I)$ of candidate solutions for each instance $I \in D_P$, and

(iii) a function m_P that assigns a positive rational number $m_P(I, x)$ called the solution value for x to each instance $I \in D_P$ and each candidate solution $x \in S_P(I)$.

Thus, an optimal solution for an instance $I \in D_P$ is a candidate solution $x^* \in S_P(I)$ such that, for all $x \in S_P(I)$, $m_P(I, x^*) \leq m_P(I, x)$ if P is a minimization problem, and $m_P(I, x^*) \geq m_P(I, x)$ if P is a maximization problem.

Due to the fact that the set of candidate solutions is finite, an algorithm for finding an optimum solution always exists. This algorithm, referred to as *exhaustive search*, simply evaluates and compares $m_P(I, x)$ for all $x \in S_P(I)$. Unfortunately, the search space of many combinatorial problems grows exponentially with the problem size, i.e., the number of components in a solution vector x. Thus, this complete enumeration scheme becomes impractical. For a large class of combinatorial optimization problems no alternative algorithms running in polynomial time are known. This phenomenon has led to the development of complexity theory [113], and in particular, to the theory of $\mathcal{NP}$-completeness.

2.3 Computational Complexity

The theory of $\mathcal{NP}$-completeness is focused on decision problems such as:

Is there a feasible solution $x \in S_P(I)$ such that $m_P(I, x) \le L$ ($m_P(I, x) \ge L$) ?

Two basic classes of decision problems are distinguished: the class $\mathcal{P}$ of decision problems that can be solved by a polynomial-time algorithm, and the class $\mathcal{NP}$ of decision problems that can be solved by a non-deterministic polynomial-time algorithm. The latter consists of two stages. In the first stage, a solution to a given instance I is *guessed*. In the second stage, this solution is *checked* by a deterministic polynomial verification algorithm [113].

Given these two classes, $\mathcal{NP}$-complete problems can be defined [113, 159]:

Def. 2.1 *A decision problem p is $\mathcal{NP}$-complete, if (a) $p \in \mathcal{NP}$, and (b) all problems in $\mathcal{NP}$ can be reduced to p by a polynomial-time algorithm.*

Probably the most important open question in computer science is whether $\mathcal{P} = \mathcal{NP}$. From the above definition follows immediately that if for one problem in $\mathcal{NP}$ a polynomial time algorithm can be found, all problems in $\mathcal{NP}$ can be solved in polynomial time and thus $\mathcal{P} = \mathcal{NP}$. However, it is commonly believed that $\mathcal{P} \neq \mathcal{NP}$, but no proof has been found yet.

So far, nothing has been said about combinatorial optimization problems. Optimization problems cannot be $\mathcal{NP}$-complete, since they are not decision problems, even though for each optimization problem a decision problem can be defined which is equivalent in complexity. However, the notion of $\mathcal{NP}$-hard problems is less restricted than the definition of $\mathcal{NP}$-completeness [159]:

Def. 2.2 *A problem (decision or otherwise) is $\mathcal{NP}$-hard if all problems in $\mathcal{NP}$ are polynomially reducible to it.*

This definition includes decision problems that are not contained in $\mathcal{NP}$ as well as problems which are not decision problems.

This work concentrates on $\mathcal{NP}$-hard combinatorial optimization problems for which the equivalent decision problems exist which are $\mathcal{NP}$-complete. Due to their computational complexity (assuming $\mathcal{P} \neq \mathcal{NP}$), powerful approximation algorithms are required that – although they do not guarantee to find the optimum solution – are able to find optimum or near–optimum solutions in short time.

2.4 Examples of COPs

Some combinatorial optimization problems that have been studied extensively are presented in the following paragraphs. The problems are selected according to their importance to the development of heuristics. They constitute only a small fraction of the family of COPs and will be studied in detail in the following chapters. A virtually unlimited number of other combinatorial optimization problems exists, which can not be considered in this work.

2.4.1 The Traveling Salesman Problem

The *traveling salesman problem* (TSP)[1] is a well–known COP, and has attracted many researchers from various fields, partly because it is hard to solve but can be easily stated: given a set of n cities and the geographical distance between them, the traveling salesman has to find the shortest tour in which he visits all the cities exactly once and returns to his starting point. More formally, the tour length

$$l(\pi) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)} \tag{2.1}$$

has to be minimized, where d_{ij} is the distance between city i and city j and π a permutation of $\langle 1, 2, \ldots, n\rangle$. Thus, an instance $I = \langle D \rangle$ is defined by a distance matrix $D = (d)_{ij}$, and a solution (TSP tour) is a vector π with $j = \pi(i)$ denoting city j to visit at the i-th step.

A special case of the TSP is the Euclidean TSP. Here, the distance matrix d_{ij} is symmetric, that is $d_{ij} = d_{ji} \quad \forall\, i, j \in \{1, 2, \ldots, n\}$, and the triangle inequality holds: $d_{ij} \leq d_{ik} + d_{kj} \quad \forall\, i, j, k \in \{1, 2, \ldots, n\}$. The distance between two cities is defined by the Euclidean distance between two points in the plane. These two assumptions do not lead to a reduction of the complexity, hence the problem remains $\mathcal{NP}$-hard.

Although there is a TSP tour for every permutation π, there are many permutations that represent the same tour. Alternatively to equation (2.1), the TSP can be defined as finding a shortest Hamiltonian cycle in a complete weighted graph $G = (V, E, d)$ where the set $V = \{1, \ldots, n\}$ represents the cities and the edge set E the arcs between them. A weight that corresponds to the distance d_{ij} between the incident cities is assigned to each edge. Thus a solution T is a subset of E with $|T| = |V| = n$.

Let the matrix $X = (x_{ij})$ be a boolean matrix with $x_{ij} = 1$, if the arc from i to j is in the tour, 0 otherwise. The TSP is then defined as:

$$\text{minimize } l(X) = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}\, x_{ij} \tag{2.2}$$

$$\text{subject to} \quad \sum_{i=1, i\neq j}^{n} x_{ij} = 1 \quad \forall\, j \in V \tag{2.3}$$

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1 \quad \forall\, i \in V \tag{2.4}$$

$$\sum_{i\in Q}\sum_{j\in V-Q} x_{ij} \geq 1 \quad \forall\, Q \subset V \tag{2.5}$$

[1]The TSP was formerly known under the name *traveling salesperson problem*. In 1976, researchers of the field have agreed to use the term traveling salesman [186].

The above equations illustrate that the TSP can be formulated as a zero/one integer programming problem. While the first two constraints ensure that the degree of each node is two (one incoming and one outgoing edge), the last constraint ensures that solutions are not consisting of disjoint sub-tours. In Figure 2.1(b), a graph is given that does not represent

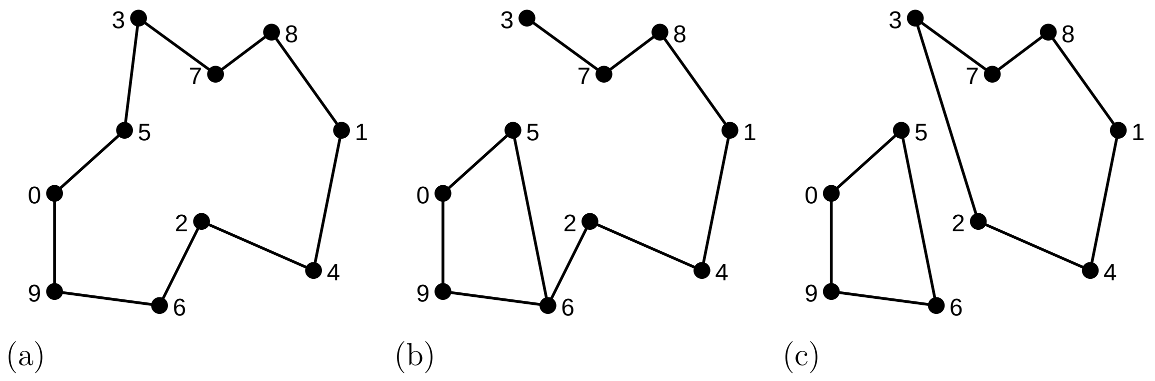


Figure 2.1: A feasible and two infeasible traveling salesman tours

a TSP tour, since node 3 and node 6 do not have degree 2, hence the constraints (2.3,2.4) are not obeyed. Figure 2.1(c) shows the violation of constraint equation (2.5): although all nodes have degree two, there are two disjoint sub-tours and thus the solution becomes infeasible. The elimination of the constraint (2.5) leads to the well–known *assignment problem*, which can be solved in polynomial time by the Hungarian method [267].

Applications of the TSP

The TSP has its applications in a wide range of domains, including drilling of printed circuit boards (PCBs) of 17000 nodes (cities) and more [193, 261], X–ray crystallography with up to 14000 nodes [33], VLSI–chip fabrication with as many as 1.2 million nodes [177], as well as overhauling of gas turbine engines of aircrafts [251], mask plotting in PCB production [128, 178], computer wiring and clustering of data arrays [187], scheduling, seriation in archaeology, and the control of robots [261].

2.4.2 Graph Partitioning

The *graph partitioning problem* (GPP) is an $\mathcal{NP}$-hard combinatorial optimization problem [113]. Given an undirected graph $G = (V, E)$, the GPP is to find a partition of the nodes in k sets of equal size, denoted $V_1, V_2, \ldots V_k$, so that the number of edges between nodes in different sets is minimized. More formally, the problem is to minimize

$$c(V_1, \ldots, V_k) = |e(V_1, \ldots, V_k)|, \text{ with} \tag{2.6}$$

$$e(V_1, \ldots, V_k) = \{(i, j) \in E | \exists\, l \in \{1, \ldots, k\} : i \in V_l \wedge j \notin V_l\}, \tag{2.7}$$

where $c(\cdot)$ is referred to as the cut size of the partition and $e(\cdot) \in E$ is referred to as the (edge) cut. An instance $I = \langle A \rangle$ of the GPP consists of an adjacency matrix A of the graph G.

A special case of the GPP is the *graph bipartitioning problem (GBP)* (also known as the *graph bi-section problem*) which can be defined as minimizing

$$c(V_1, V_2) = |e(V_1, V_2)|, \text{ with } \quad e(V_1, V_2) = \{(i, j) \in E | i \in V_1 \wedge j \in V_2\}. \tag{2.8}$$

Figure 2.2 displays a partition of a small geometric graph into two sets. The edges in the cut are emphasized.

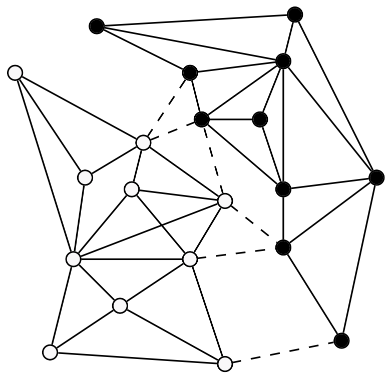


Figure 2.2: A solution to a graph bipartitioning problem

The GBP can be formulated as a 0/1 integer programming problem as follows.

$$\text{minimize } c(x) \quad = \quad \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}\, x_i\, (1 - x_j) \tag{2.9}$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_i = \frac{n}{2}, \tag{2.10}$$

where matrix $A = (a_{ij})$ is the adjacency matrix of the graph G. A value of 0 in the solution vector x for component x_k denotes that vertex k belongs to the first set V_1, and to V_2 otherwise.

If it is not required that both sets have the same size (equation (2.10)), the optimum can be found in polynomial time by the algorithm of Ford and Fulkerson [102]. This problem is known as the *minimum cut problem.*

Applications of the GPP

The graph partitioning problem arises in many applications such as parallel and distributed computing, VLSI circuit design and simulation, transportation management, and data mining [252, 276].

2.4.3 The Quadratic Assignment Problem

The *quadratic assignment problem (QAP)* has been introduced by Koopmans and Beckmann [176] to describe a location problem where a set of facilities has to be assigned to given locations at minimal cost. Mathematically, the QAP can be defined as follows.

Given two $n \times n$ matrices $A = (a_{ij})$ and $B = (b_{ij})$, the following cost function for assigning n facilities to n locations has to be minimized:

$$C(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}\, b_{\pi(i)\pi(j)}, \tag{2.11}$$

where a_{ij} denotes the distance between location i and location j, and b_{kl} denotes the flow of materials from facility k to facility l. Thus, a QAP instance $I = \langle A, B \rangle$ consists of a flow and distance matrix, and a solution π denotes a permutation of $\langle 1, 2, \ldots, n \rangle$ ($\pi(i)$ denotes the facility assigned to location i).

As an alternative to equation (2.11), the QAP can be formulated as a 0/1 integer programming problem as follows:

$$\begin{aligned} \text{minimize } C(X) \;&=\; \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} a_{ik}\, b_{jl}\, x_{ij}\, x_{kl}, \qquad (2.12)\\ \text{subject to} \quad & \sum_{i=1}^{n} x_{ij} = 1, \quad \forall\, j = 1, 2, \ldots, n,\\ & \sum_{j=1}^{n} x_{ij} = 1, \quad \forall\, i = 1, 2, \ldots, n,\\ & x_{ij} \in \{0, 1\}, \quad \forall\, i, j = 1, 2, \ldots, n. \end{aligned}$$

This definition has some resemblance to the 0/1 integer programming definition of the TSP. In fact, both TSP and GPP are special cases of the QAP, as shown below.

Applications of the QAP

The QAP has many practical applications, such as backboard wiring on electronic circuits [285] or the design of typewriter keyboards and control panels [51, 207]. Furthermore, it has been used in facility location problems, in particular hospital planning [182, 85] and in finding locations for new buildings of a campus [73]. Besides other domains of engineering and design [114], a new application of the QAP in biology has recently been discovered in the context of *indirect gradient analysis* (reconstruction of the intensity of some latent environmental factors from species' responses) [52].

Two Special Cases of the QAP

The TSP can be formulated as a QAP by defining the matrix $A = (a_{ij})$ and $B = (b_{ij})$ as follows. Let

$$a_{ij} = \begin{cases} 1 & \text{if} \quad j = (i + 1 \bmod n) \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad b_{ij} = d_{ij}, \tag{2.13}$$

with d_{ij} denoting the distance between city i and city j. Then

$$C(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}\, b_{\pi(i)\pi(j)} = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)}, \tag{2.14}$$

which is, in fact, the definition of the TSP. A permutation π provides a tour through n cities where the city $\pi(i)$ is the city to visit at the i-th step.

The graph bipartitioning problem, in which a partition of a graph $G = (V, E)$ into two equally sized sets with a minimum number of edges between the different sets is desired, can be expressed as a QAP by defining

$$a_{ij} = \begin{cases} 0 & \text{if} \quad i,j \le \frac{n}{2} \vee i,j > \frac{n}{2} \\ 1 & \text{otherwise} \end{cases} \quad \text{and} \quad b_{ij} = \begin{cases} 1 & \text{if} \quad (i,j) \in E \\ 0 & \text{otherwise} \end{cases} . \tag{2.15}$$

A partition represented by a permutation π is defined as follows. A vertex j belongs to the first set, if $\pi(j) \le \frac{n}{2}$, and to the second set otherwise.

2.4.4 Vehicle Routing

The *vehicle routing problem* (VRP) arises in distribution management. The problem is to determine the optimal delivery route for vehicles through a set of customer locations, subject to some constraints. More formally, the VRP can be defined on a weighted graph $G = (V, E, d)$ with $V = \{0, 1, \ldots, n\}$ denoting the set of customer locations including the depot (vertex 0), E the set of all arcs between the vertices, and a distance/travel time d assigned to each arc in the graph. The total travel time of m vehicles starting from the depot is to be minimized:

$$\text{minimize } l(\pi) \quad \sum_{j=1}^{m} \sum_{i=1}^{l_j - 1} d_{\pi_j(i),\pi_j(i+1)} + d_{\pi_j(l_j),0} + d_{0,\pi_j(1)} \tag{2.16}$$

$$\text{subject to} \quad \sum_{i=1}^{l_j} w_{\pi_j(i)} \le C_j \quad \forall j = 1, \ldots m, \tag{2.17}$$

where l_j denotes the number of customers for vehicle j to visit, w_i denotes the weight of customer i's demand, and the permutation π_j of a subset defines the route of vehicle j ($\pi_j(i)$ denotes the i-th customer in the route of vehicle j). Each customer has to be served by exactly one vehicle.

Several variants of the VRP exist, such as (a) the *capacitated VRP* in which the total weight of any route may not exceed the vehicle capacity C_j as defined in equation (2.17); (b) the *time constrained VRP*, in which each customer has a service time s_i and the total duration of any route may not exceed a time limit L; (c) the *VRP with time windows*, in which each customer must be visited in a given time interval and the vehicles are allowed to wait if they arrive before the beginning of the time window. Alternatively, the number of vehicles may be minimized instead of the total delivery time.

An instance of the capacitated VRP consists of a distance matrix $D = d_{ij}$, a weight vector w, and the capacity vector C ($I = \langle D, w, C \rangle$). A solution to a VRP is displayed in Figure 2.3. The TSP is a special case of the uncapacitated VRP: in the TSP only one vehicle is used to visit the customers/cities.

2.4.5 Scheduling

Given n jobs and m machines, the *job shop scheduling problem* (JSSP) is defined as follows. Each job consists of a sequence of operations (tasks), which must be executed in a given order. Furthermore, each operation has to be executed on a given machine for a given period of time. Assuming that a machine can perform at most one operation at a time, the problem

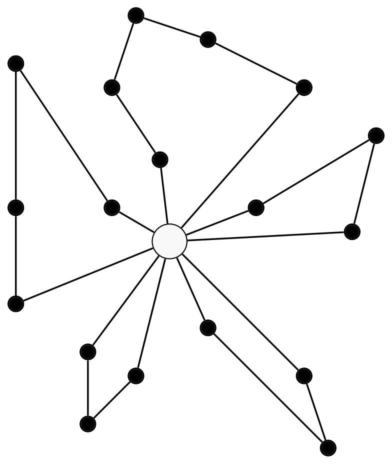


Figure 2.3: A solution of a vehicle routing problem

is to find a schedule, i.e. an assignment of operations to time intervals, such that the total length of the schedule (called the makespan) is minimal. More formally, let J, M and O be the sets of jobs, machines, and operations, respectively. For each operation $o \in O$, there is a job $j(o) \in J$ to which it belongs, a machine $m(o) \in M$ on which it must be processed, and a processing time $t_p(o)$. Furthermore, for each $o \in O$, its successor in the job is denoted $\text{succ}(o) \in O$. The problem is to find a start time t_s for each operation $o \in O$, and can be defined as:

$$\text{minimize} \quad \max_{o \in O} (t_s(o) + t_p(o)) \tag{2.18}$$

$$\text{subject to} \quad t_s(\text{succ}(o)) \geq t_s(o) - t_p(o) \quad \forall o \in O, \tag{2.19}$$

$$t_s(o') \geq t_s(o) + t_p(o) \ \vee \ t_s(o) \geq t_s(o') + t_p(o') \tag{2.20}$$

$$\forall o, o' \in O \mid o \neq o' \ \wedge \ m(o) = m(o'). \tag{2.21}$$

According to Roy and Sussmann [265], an instance $I = \langle J, O, M, t_p \rangle$ of the job–shop scheduling problem can be represented by a vertex weighted disjunctive graph $G = (V, A, E)$, where the vertex set V corresponds to the set of operations O, the arc set A consists of arc connecting consecutive operations of the same job, and the edge set E consists of edges connecting operations that must be executed on the same machine. An example of a solution to a 3×3 instance is displayed in Figure 2.4.

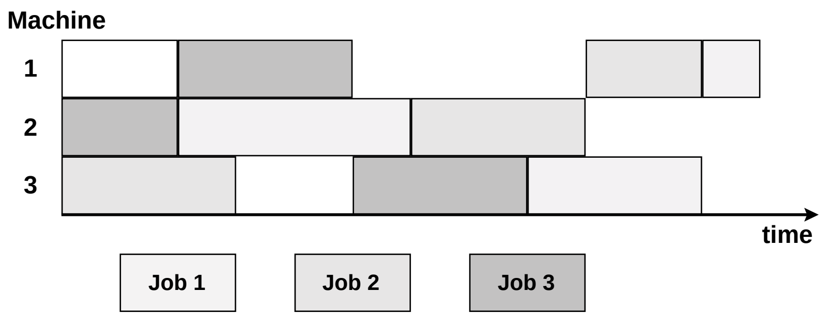


Figure 2.4: A solution to a 3-jobs/3-machines instance of the JSSP

Variants of the JSSP

In the *flow shop scheduling problem* (FSSP), there is a strict ordering in which the operations have to be performed and this ordering is the same for each job, e.g. first do the machine 1 operation, then the machine 2 operation, and so on. In the *general job shop scheduling problem* (GJSSP), the operations within a job must be partially ordered. If there is no order required in the processing of the operations within a job, the problem is called *open shop scheduling problem* (OSSP).

Applications of Scheduling Problems

Scheduling problems occur wherever a number of tasks has to be performed with limited resources [58]. Thus, applications can be found in production planning, project resource management, and distributed or parallel computing.

2.4.6 Unconstrained Binary Quadratic Programming

In the *unconstrained binary quadratic programming problem* (BQP), a symmetric rational $n \times n$ matrix $Q = (q_{ij})$ is given, and a binary vector of length n is searched for, such that the quantity

$$f(x) = x^t\, Q\, x = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij}\, x_i\, x_j, \quad x_i \in \{0,1\}\ \forall\, i = 1, \ldots, n \tag{2.22}$$

is maximized. This problem is also known as the *(unconstrained) quadratic bivalent programming problem*, *(unconstrained) quadratic zero–one programming problem*, or *(unconstrained) quadratic (pseudo-) boolean programming problem* [26]. The general BQP is known to be $\mathcal{NP}$-hard but there are special cases that are solvable in polynomial time [26].

Applications of the BQP

The BQP has a large number of applications, for example in capital budgeting and financial analysis problems [185, 205], CAD problems [182], traffic message management problems [109], machine scheduling [4], and molecular conformation [250]. Furthermore, several other combinatorial optimization problems can be formulated as a BQP, such as the maximum cut problem, the maximum clique problem, the maximum vertex packing problem and the maximum independent set problem [151, 245, 246].

Special Cases of the BQP

The BQP has been shown to be a generalization of other combinatorial optimization problems. For example, the maximum clique problem and the maximum independent set problem are known to be special cases of the BQP. Let $G = (V, E)$ be an undirected graph and $\overline{G} = (V, \overline{E})$ be the complement graph of G, where $\overline{E} = \{(i,j) \,|\, i,j \in V,\ i \neq j, \text{ and}(i,j) \notin E\}$. Furthermore, let $A_G = (a_{ij})$ be the adjacency matrix of G, I denote the identity matrix, and $T(x) = \{i \,|\, x_i = 1,\ i \in V\}$. Then, the *maximum clique problem* is

$$min_{x \in X}\, f(x) = x^t Q x, \quad \text{where} \quad Q = A_{\overline{G}} - I. \tag{2.23}$$

If x^* solves equation (2.23), the maximum clique of G is defined as $C = T(x^*)$ with $|C| = f(x^*)$.

Similarly, the *maximum independent set problem* is

$$min_{x \in X} \, f(x) = x^t Q x, \quad \text{where} \quad Q = A - I. \tag{2.24}$$

If x^* solves equation (2.24), the maximum independent set of G is defined as $S = T(x^*)$ with $|S| = f(x^*)$.

In the *maximum cut problem*, the objective function

$$c(x) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \, x_i \, (1 - x_j) \tag{2.25}$$

has to be maximized, where w_{ij} denotes the weight of the edge $(i, j) \in E$ in the graph $G = (E, V)$ for which the maximum cut is desired. The maximum cut problem can be formulated as a 0/1 quadratic programming problem by assigning:

$$q_{ij} = -\frac{1}{2} w_{ij}, \;\; \forall i \neq j, \quad \text{and} \quad q_{ii} = \frac{1}{2} \sum_{j=1}^{n} w_{ij}, \;\; \forall i. \tag{2.26}$$

The maximum cut size $c(x^*)$ is equal to the objective $f(x^*)$ of the corresponding BQP, and the cut itself is $C = \{(i, j) \in E \mid x_i^* = 0 \text{ and } x_j^* = 1\}$.

Another application of the BQP arises in condensed matter physics. The calculation of ground states in *Ising Spin Glasses* is a combinatorial optimization problem in which a configuration of the spins with minimum energy is searched. The energy of an Ising spin glass, in which the spins lie on a two dimensional grid, is given by the Hamiltonian

$$H(\omega) = -\sum_{i} \sum_{j} J_{ij} \, s_i s_j, \quad s_i, s_j = \pm 1, \tag{2.27}$$

where J_{ij} denotes the interaction between site i and j on the grid. By setting

$$q_{ij} = 4 J_{ij}, \;\; \forall i \neq j, \quad \text{and} \quad q_{ii} = -4 \sum_{j=1}^{n} J_{ij}, \;\; \forall i, \tag{2.28}$$

the solution of the BQP yields a configuration with minimum energy, where $s_i = 2x_i - 1 \; \forall i$ and $H(\omega) = -f(x) - \sum_i \sum_j J_{ij}$.

2.4.7 *NK*-Landscapes

To study rugged fitness landscapes, Kauffman [168, 169] developed a formal model for gene interaction which is called the *NK*-model. In this model, N refers to the number of parts in the system, i.e. genes in a genotype or amino acids in a protein. Each part makes a fitness contribution which depends on the part itself and K other parts. Thus, K reflects how richly cross-coupled the system is; it measures the richness of interactions among the components of the system, called epistasis.

Each point in the *NK*-fitness landscape is represented by a bit string of length N and can be viewed as a vertex in the N-dimensional hypercube. The fitness f of a point $b = b_1, \ldots, b_N$ is defined as follows:

$$f(b) = \frac{1}{N} \sum_{i=1}^{N} f_i(b_i, b_{i_1}, \ldots, b_{i_K}), \tag{2.29}$$

where the fitness contribution f_i of the gene at locus i depends on the allele (value of the gene) b_i and K other alleles $b_{i_1}, \dots, b_{i_K}$. The function $f_i : \{0,1\}^{K+1} \rightarrow I\!R$ assigns a uniformly distributed random number between 0 and 1 to each of its 2^{K+1} inputs. The values for $i_1, \dots, i_K$ are chosen randomly from $\{1, \dots, N\}$ or from the left and right of locus i.

With this model, the "ruggedness" of a fitness landscape can be tuned by changing the value of K and thus the number of interacting genes per locus. Low values of K indicate low epistasis and high values of K indicate high epistasis.

From the viewpoint of combinatorial optimization, the *NK*-model represents an unconstrained binary programming problem that can be seen as a generalization of the BQP.

2.4.8 The Knapsack Problem

The *knapsack problem* (KP) is another well-known combinatorial problem. Given a knapsack which can be used to transport a number of items with a maximum total weight, the task is to select a subset of all items available such that the value of the items is maximized. More formally, the KP can be defined as follows:

$$\text{maximize} \quad \sum_{i \in K} c_i \tag{2.30}$$

$$\text{subject to} \quad \sum_{i \in K}^{n} w_i \leq W, \quad K \subset \{1, \dots, n\}, \tag{2.31}$$

where c_i denotes the value (or resulting profit) of item i, w_i the weight of item i, and W the total weight the knapsack can hold. An instance of the knapsack problem is hence defined by the tuple $I = \langle c, w, W \rangle$ with $c = (c_1, \dots, c_n) \in I\!R^n$, $w = (w_1, \dots, w_n) \in I\!R^n$ and $W \in I\!R$.

An generalization of this problem is the *multidimensional knapsack problem* (MKP):

$$\text{maximize} \quad \sum_{i \in K} c_i \tag{2.32}$$

$$\text{subject to} \quad \sum_{i \in K} w_{ij} \leq W_j, \quad K \subset \{1, \dots, n\}, \quad \forall j = 1, \dots m. \tag{2.33}$$

Here, the "weight" of an item as well as the maximum capacity of the knapsack has m dimensions. Formulated as 0/1 integer programming problem, the MKP is:

$$\text{maximize} \quad \sum_{i=1}^{n} c_i \, x_i \tag{2.34}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_{ij} \, x_i \leq W_j, \quad \forall j = 1, \dots m. \tag{2.35}$$

The KP is one of the simplest constrained 0/1 integer programming (binary programming) problems since without its constraints it is a linear problem that can be solved in $O(n)$.

Applications of the MKP

The (multidimensional) knapsack problem has its application in capital budgeting, project selection and capital investment, budget control, and numerous loading problems [267].

2.5 Exact Methods for Solving COPs

The simplest way to obtain optimum solutions to combinatorial optimization problems is to evaluate all possible solutions. As mentioned above, this approach is impractical due to the large number of candidate solutions that usually grows exponentially with the problem size. However, there are other ways to find guaranteed optimum solutions.

2.5.1 Branch & Bound

An algorithm for finding optimal solutions consists of methods for finding lower and upper bounds for the optimal solution and an enumeration scheme. Assuming a minimization problem, upper bounds are usually obtained by effective heuristics that produce near optimum solutions in short time. Lower bounds are obtained by relaxations of the problem by removing one or more constraints. The enumeration scheme works as follows. In each stage, the problem is split into subproblems such that the union of feasible solutions of the subproblems gives the feasible solutions of the master problem. Subproblems are further divided into subproblems until they are solved, i.e. their lower bounds are equal to their upper bounds, or their lower bounds are above the best feasible solution found so far. Thus, the approach produces a *branching tree* in which each node corresponds to a problem and the sons of the node represent the subproblems into which it is split. To prevent an exponential growth of the nodes in the tree, relaxations for producing strong lower bounds and good heuristics for producing upper bounds are required.

In the *branch & bound* approach, discrete relaxation schemes are used for combinatorial optimization problems to provide lower bounds.

The branch & bound approach has been applied successfully to the asymmetric TSP (with asymmetric distance matrices) using the assignment relaxation (equation (2.5) is ignored) [92, 222]. However, for most other problems there appear to be no discrete relaxations that are strong enough for solving large problem instances.

2.5.2 Branch & Cut

An even more elaborate approach is based on *linear programming* [63, 315]. The basic idea is to find a relaxation in form of a linear program with the same optimal solution as the original problem. The linear programming problem (LP) is defined as follows:

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & A\,x \le b, \quad x \le 0 \in I\!R^n. \end{aligned} \tag{2.36}$$

For such a linear program, an efficient search algorithm called the *simplex algorithm* [62, 63] exists that provides always the optimum solution. This algorithm works by systematically searching the corners of the polytope (polyhedron) P defined by the inequalities of the LP: $P = \{x \in I\!R^n \mid A\,x \le b\}$.

Thus, to solve a COP with linear programming techniques, the search space is enlarged by extending the solution vectors (usually of a 0/1 integer formulation) to vectors of continuous variables. Since not all facets of the polytope P_C (the inequalities $a^t\,x < \alpha$) are known for every combinatorial problem or the number of facets is simply too high, a *cutting plane approach* has been developed. This approach works as follows. First, an initial polytope $P \supseteq P_C$ is generated so that the LP can be solved in reasonable time. Then, an LP solver is used to generate a solution x^*. If the solution x^* represents a feasible solution to the COP, then the algorithm terminates: the optimum solution is found. Otherwise, the algorithm searches for a cut (facet) such that x^* is cut off the polytope by ensuring that the new polytope still contains the polytope of the COP. The inequality found is added to the system of equations and the resulting LP is solved to obtain a new x^*. These steps are repeated until the optimum is found or the algorithm fails to find a new feasible cut. Since the latter case is more likely to occur, a branching rule can be used to split the problem into subproblems and the cutting plane procedure can be applied recursively to the subproblems. The resulting approach is called *branch & cut* [242].

Branch & cut has been successfully applied to various combinatorial problems such as the traveling salesman problem [91], the Steiner problem on graphs [195], the spin glass problem [272], the graph bipartitioning problem [42], and the maximum cut problem [161]. Especially for the TSP, enormous progress has been made in the last 30 years in solving large instances to optimality. Table 2.1 gives an overview of the history of solving TSP instances to optimality with branch & cut. The year of publication, the number of cities (N), the total number of candidate solutions ($|S| = (N-1)!/2$), and the names of the researchers are provided.

Year	$\boldsymbol{N}$	$\boldsymbol{\lvert S\rvert}$	**Researchers**
1954	48	$> 10^{59}$	Dantzig, Fulkerson & Johnson [64]
1980	120	$> 10^{196}$	Grötschel [126]
1980	318	$> 10^{656}$	Crowder & Padberg [60]
1987	532	$> 10^{1217}$	Padberg & Rinaldi [242]
1991	666	$> 10^{1589}$	Grötschel & Holland [127]
1991	2392	$> 10^{7041}$	Padberg & Rinaldi [243]
1992	3038	$> 10^{9259}$	Applegate, Bixby, Chvàtal & Cook [10]
1993	4461	$> 10^{14341}$	Applegate, Bixby, Chvàtal & Cook [10]
1994	7397	$> 10^{25405}$	Applegate, Bixby, Chvàtal & Cook [10]
1998	13509	$> 10^{49931}$	Applegate, Bixby, Chvàtal & Cook [11]

Table 2.1: History of Records in solving the TSP with Branch & Cut

However, finding appropriate cuts is a problem dependent and highly complicated task. For each COP, new theories are necessary for deriving useful classes of facets. In case of the TSP, much effort has been made in developing polyhedral theories [129] in the last decades, and it appears that the TSP is well suited for this kind of approach. For other COPs, however, theories are needed to apply branch & cut to instances of practical interest.

2.5.3 Heuristic Methods

Heuristics are search methods that find (near) optimum solutions to optimization problems in short time. In comparison to exact approaches, they do not guarantee to find optimum solutions nor do they generally provide a guarantee to find solutions within a certain range to the optimum.

Nevertheless, they are of great importance since they are the only way to arrive at high quality solutions for large combinatorial problems of practical interest. Many heuristics have the advantage to be applicable to a wide range of problems so that the time for developing an optimization algorithm for a new problem is usually short. These heuristics can easily be modified to account for changes in the objective function. If a previously not considered constraint is added to the problem description, heuristics can be modified easily to deal with the altered problem. Due to the complexity of combinatorial spaces, exact methods are only in rare cases an alternative to heuristics. Even in those cases in which exact methods are required, powerful heuristics are necessary to provide good upper bounds for a branching approach to be effective.

2.6 Classification of COPs

According to [74], there are at least four classes of combinatorial optimization problems. For each class, a few examples are provided:

Assignment Problems: The linear and the quadratic assignment problem are examples of this type, as well as time tabling problems (the assignment of teachers to classes and rooms).

Ordering Problems: The traveling salesman problem, the linear ordering problem, the chinese postman problem, and scheduling problems constitute this class of problems.

Partitioning Problems: The graph partitioning problem and the number partitioning problem are of this class.

Subset Selection Problems: The knapsack problem, the set partitioning problem, the set covering problem, the graph bipartitioning problem, and the maximum cut problem belong to this type.

Some COPs can hardly be classified according to these classes. Other problems belong to more than one class such as the vehicle routing problem which is a combination of a partitioning problem and an ordering problem.

Another form of classification can be derived if the constraints defined in the problems are considered. This kind of classification is important for developing heuristics. There are problems with

(i) *no constraints*, like the BQP (and its special cases) or the *NK*-model,

(ii) *implicit constraints*, like the TSP or the GBP,

(iii) *explicit constraints* like the MKP, and problems with

(iv) *implicit and explicit constraints* like the capacitated vehicle routing problem.

The difference between implicit and explicit constraints is that implicit constraints are problem instance independent. For example, the size constraints in the GBP defined in equation (2.10) (both sets are required to have the same size) do not depend on the structure of the graph and are thus instance independent, while the capacity constraints in the MKP do depend on the instance to be solved: the total capacity W_j is part of the instance description.

2.7 Summary

In this chapter, a formal definition of combinatorial optimization problems (COPs) has been provided and examples of this class of problems have been described. COPs can be distinguished from other optimization problems in that the decision variables constituting a solution vector are discrete. For example, in selection problems, a solution is usually a binary zero-one vector. For ordering problems, a permutation of a vector with components of different discrete values constitutes a solution. Some of the most famous problems in operations research have been introduced: the traveling salesman problem (TSP), the graph partitioning problem (GPP) and its special case, the graph bipartitioning problem (GBP), the quadratic assignment problem (QAP), vehicle routing problems (VRP), scheduling problems like flow shop (FSSP) and job-shop scheduling problems (JSSP), unconstrained binary programming (BQP), *NK* model of fitness landscapes stemming from biology, and the multidimensional knapsack problem (MKP). Some of the problems will be investigated in detail in the following chapters.

Furthermore, exact methods for solving COPs – in particular branch & bound and branch & cut methods – have been discussed and the importance of effective heuristics has been stressed. Although it has been shown that branch & cut methods produce remarkable results for the TSP, heuristics are still preferable to arrive at high quality solutions in short time for large practical problems and are not limited to a particular problem domain. Furthermore, they are required in exact methods to produce feasible solutions (upper or lower bounds).

A classification of COPs based on the characteristics of constraints present in a problem has been provided: implicit constraints are problem inherent and are independent of the problem instance to solve. Explicit constraints, on the other hand, depend on the problem instance description and thus vary from instance to instance. The distinction between implicit and explicit constraints is important for developing heuristics.

In the following chapter a highly effective family of hybrid heuristics is introduced, called *memetic algorithms*, which have been shown to be especially effective in combinatorial optimization.

Chapter 3

Memetic Algorithms

3.1 Introduction

Evolutionary computation, a tremendously growing field of computer science, covers all aspects of the simulation of evolutionary processes in computer systems. On the one hand, simulations of natural evolution have been used by biologists to study adaptation in changing environments to gain insight in the evolution of the complex organisms found on earth. On the other hand, it has been shown that complex optimization problems can be solved with simulated evolution. In the last decades, wide applicability has been demonstrated by successfully applying evolutionary computation techniques to various optimization problems in the fields of engineering, management science, biology, chemistry, physics and computer science.

However, it has been shown that some kind of domain knowledge has to be incorporated into evolutionary algorithms to be competitive with other domain specific optimization techniques. There are many ways to achieve this. A promising approach is the hybridization with other (domain-specific) heuristics for the optimization problem to be solved. The resulting hybrid evolutionary algorithms often fall into the category of memetic algorithms. These algorithms are similar to traditional evolutionary algorithms, although they have more in common with principles found in the evolution of the human culture rather than in biological evolution.

This chapter is devoted to the fundamentals of evolution as well as their simulation in computer experiments with emphasis on solving optimization problems. First, genetic evolution and its simulation in evolutionary algorithms is described. Afterwards, modern alternative heuristics for solving combinatorial optimization problems are presented: some that are also biologically inspired and others that are particularly useful for incorporation into evolutionary algorithms. Finally, memetic algorithms are described in detail and the analogy to cultural evolution is shown.

3.2 Evolutionary Algorithms

Inspired by the power of natural evolution, several computer scientists independently studied evolutionary systems keeping in mind the idea that engineering problems could be solved by simulating natural evolution processes. Several evolutionary algorithms (EAs) – for example *evolution strategies*, *evolutionary programming*, and *genetic algorithms* – have been proposed since the early 1960s in which a population of candidate solutions is evolved subject to replication, variation, and selection.

Before evolutionary algorithms are described in detail, a short introduction to natural evolution is presented and the necessary biological terminology is introduced.

3.2.1 Natural Evolution

In his book *The Origin of Species* [65], Darwin presented a theory for the existence and evolution of life on earth. According to his theory, evolution is based on three fundamental concepts: *replication*, *variation*, and *natural selection*. New organisms cannot evolve solely with replication, since the produced offspring are identical copies. But due to errors in the replication process, variation is introduced that gives rise to the gradual development of new organisms. Sexual recombination is another form of variation and is itself a product of evolution. Due to the limited resources on earth, replication can not go on infinitely; individuals of the same or other species have to compete with each other and only the fittest survive. Thus, natural evolution implicitly causes the adaptation of life forms to their environment since only the fittest have a chance to reproduce ("survival of the fittest").

Natural evolution can be thought as being a gigantic optimization process in which the fitness of the species is maximized. However, it is an open-ended dynamic process in which the fitness of an individual can only be defined in relation to the environment. For example, a polar bear has a high fitness in its native environment, since it is well adapted to the cold temperatures. Bringing the polar bear to the African savanna would certainly reduce its fitness. Sometimes, species become extinct when they are not able to react to rapid changes in their environment.

From the information science point of view, natural evolution can be regarded as a huge information processing system. Each organism carries its genetic information referred to as the *genotype*. The organism's traits, which are developed while the organism grows up, constitute the *phenotype*. The genetic information is eventually passed on to the next generation if the organism reproduces before it dies. Thus, the organisms can be regarded as the mortal survival machines of the potentially immortal genetic information. While replication combined with variation allows for improving the genetic information, natural selection implicitly evaluates the fitness of each phenotype and thus indirectly of the genotype, which can be thought of as the construction plan of an organism.

Genetics

All living organisms consist of cells, and each cell contains a copy of a set of one or more *chromosomes*, which are strings of DNA (*desoxyribonucleic acid*). The chromosomes serve as a "blueprint" (construction plan) for the organism, and can be conceptually divided into *genes*. Genes are functional blocks of DNA and each encodes a particular protein. Each gene is located at a particular *locus* on the chromosome. In a very simplified model, we can think of a gene as encoding a *trait* such as eye color. The different possible settings of a trait are called *alleles*. Many complex organisms have more than a single chromosome in each cell. All chromosomes taken together and thus the complete collection of genetic material is called *genome*. The genotype mentioned above refers to the set of genes in a particular genome and gives rise to the phenotype of the organism under fetal or later development.

Two forms of reproduction can be found in nature. The first form is asexual reproduction in which an organism reproduces itself by cell division and the replication of its chromosomes. *Mutation* eventually occurs during this process: one or more alleles of genes are changed, genes are deleted, or they are reinserted at other loci on the chromosomes. In sexual recombination, the second form of reproduction, genes are exchanged between the chromosomes of the two parents to form a new set of chromosomes. This *recombination* of genetic material can be thought of as crossing-over of the chromosomes.

The *fitness* of an organism is defined as the probability that the organism will live to reproduce, called *viability*, or defined by the number of offspring the organism has, called *fertility*.

The results of genetic variations occurring during mutation or recombination are hardly predictable due to the universal effects of gene interaction called *epistasis*. *Pleiotropy* is the effect that a single gene may simultaneously affect several phenotypic traits. On the other hand, a single phenotypic characteristic may be determined by the simultaneous interaction of many genes. This effect is called *polygeny*. In Figure 3.1, pleiotropy and polygeny are illustrated. There are no one-gene, one-trait relationships in natural evolved systems.

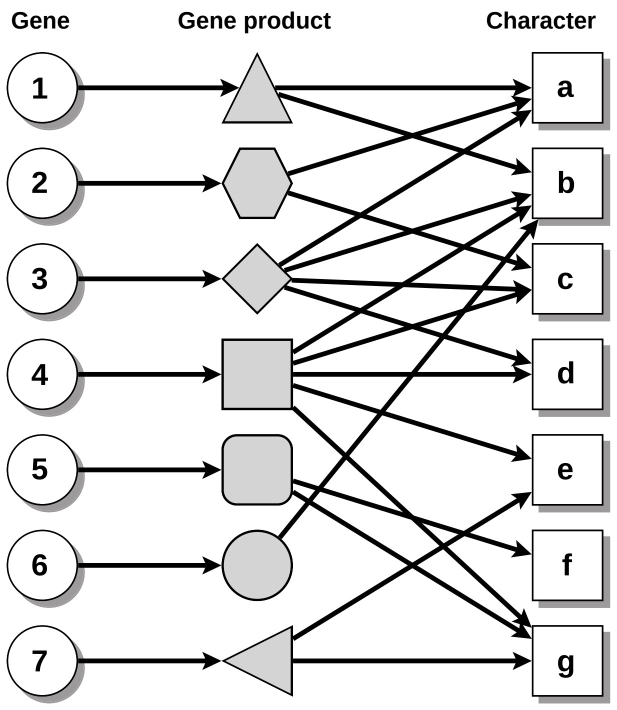


Figure 3.1: Epistatic Gene Interaction: An Example of Pleiotropy and Polygeny

Epistatic interactions in form of pleiotropy and polygeny are almost always found in living organisms so that the phenotype varies as a complex, nonlinear function of the interaction between the underlying genetic structures and the environmental conditions.

3.2.2 History of Evolutionary Computation

In the 1960s at the Technical University of Berlin, Rechenberg and Schwefel [258, 257] introduced *evolution strategies* (ES – Evolutionsstrategie in the German original), an approach they developed to optimize the real-valued parameters for devices such as airfoils. Mainly focusing on continuous parameter optimization, this idea was further developed by Schwefel [269], and is still an active area of research. In the beginning, ES included mutation and selection on a two-membered population. Later, it has been extended by allowing more than two members and alternative selection strategies. The current ES include multi-parent recombination and the self-adaptation of strategy parameters.

The technique called *evolutionary programming* (EP) was developed at the same time by Fogel, Owens, and Walsh [97, 98]. Initially, they studied a system that shows intelligent behavior by predicting its environment and producing suitable responses in the light of a given goal. Therefore, they developed an evolutionary algorithm based on mutation and selection on finite state machines. Later, EP has been applied successfully to various other problems and is also still an area of active research. ES and EP have many similarities.

Genetic algorithms (GAs) were invented by Holland [142] at the University of Michigan in the 1960s. Hollands original goal was to study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms might be imported into computer systems. Thus, GAs served as an abstraction of biological evolution. Holland provided a theoretical framework for adaptation under the GA. In his model, the chromosomes or genomes were strings of ones and zeros (bits) to which genetics-inspired operators of crossover, mutation, and inversion where applied, followed by a kind of natural selection. Hence, the genes in his model are bits with the alleles "0" and "1". Compared to ES and EP, Holland's GA was the first algorithm incorporating a form of recombination (crossover).

In the last several years, there has been an enormous amount of research in evolutionary computation with increasing interaction among the researchers of the various methods. The boundaries between GAs, EP and ES have been broken down to some extent and evolutionary algorithms have been developed that combine the advantages of the approaches. The fields of applications of EAs have been drastically extended including the evolution of computer programs known under the name *genetic programming* [179, 180], or the implementation of machine learning in *classifier systems* [142, 37]. Other extensions to the basic concepts have been made such as co-evolution [138, 247] or the hybridization of traditional problem-specific methods with EAs [70, 221].

3.2.3 Outline of Evolutionary Algorithms

Without referring to a particular algorithm, a general template of an EA is shown in Figure 3.2. All proposed methods, GAs, EP and ES are special cases of this scheme. First, an initial population is created randomly, usually with no fitness or structural bias. Then, in the main loop, a temporary population is selected from the current population utilizing a selection strategy. Afterwards, the evolutionary operators mutation and/or recombination are applied to some or all members (individuals) of the temporary population. Usually, the main loop is repeated until a termination criterion is fulfilled (a time limit is reached or the number of generations evolved exceeds a predefined limit). The newly created individuals are evaluated by calculating their fitness. Before a new generation is processed, the new population is selected from the old and the temporary population. Now, the algorithm can continue by building a new temporary population. Besides the way the methods encode the candidate solutions of the problem to solve, they differ in the order and rate in which the variation operators are applied and in the type of selection strategy they use.

Fitness Evaluation

The fitness evaluation is the central part of an evolutionary algorithm. The fitness function is usually the objective function of the problem to be solved by the evolutionary algorithm.

```
procedure EA;
  begin
    $t := 0$;
    initializePopulation($P(0)$);
    evaluate($P(0)$);
    repeat
      $P'$ := selectForVariation($P(t)$);
      recombine($P'$);
      mutate($P'$);
      evaluate($P'$);
      $P(t+1)$ := selectForSurvival($P(t)$, $P'$);
      $t := t + 1$;
    until terminate = true;
  end;
```

Figure 3.2: The EA pseudo code

Thus, for each problem, the fitness function has to defined individually.

Constrained optimization problems have to be treated differently from unconstrained problems. For example, before the fitness of a solution is evaluated, a repair algorithm has to be applied to transform infeasible to feasible solutions if the evolutionary variation operators do not always produce feasible solutions. An alternative approach to repairing is the use of penalty functions. In this approach, a penalty term is added to the fitness function to penalize infeasible solutions in such a way that the EA is focused on the feasible region of the search space.

For many problems, the fitness evaluation dominates the running time of the algorithm. However, problem specific characteristics may be exploited to reduce the running time of the fitness evaluation. Furthermore, fitness evaluations can be performed in parallel on workstations with more than one processor or in workstation clusters.

Selection

Two forms of selection can be found in evolutionary algorithms, as shown in Figure 3.2. In the first (selectForVariation), individuals are chosen for recombination and/or mutation. In the second (selectForSurvival), the indivduals for the new generation are selected. The latter is sometimes called *replacement*, since some or all parents are replaced by some or all of the offspring.
Strategies for **selection for variation** can be divided as follows:

Fitness-proportionate selection:
In *fitness-proportionate selection*, the probability of selecting individual s_i is given by:

$$p(s_i) = \frac{f(s_i)}{\sum_{s_j \in P} f(s_j)}. \tag{3.1}$$

Fitness-proportionate selection can be realized by *roulette wheel sampling* [120]. Roulette wheel sampling works by spinning a roulette wheel, on which each individual has a roulette wheel slot sized in proportion to its fitness.

Rank-based selection:
A drawback of the fitness-proportionate selection method is that with decreasing variance of the fitness values of the population, the sampling becomes purely random. Therefore, *rank-based selection* [17] has been proposed to keep selection pressure constant independent of the variance of the fitness values. In the *linear ranking* model, the probability of selecting individual s_i is given by the formula

$$p(s_i) = p_{\max} - (p_{\max} - p_{\min})\frac{i-1}{n-1}, \tag{3.2}$$

where n denotes the population size, and $p_{\min}$ and $p_{\max}$ denote the minimum and maximum selection probability. The latter are parameters of the method. Alternatively, non-linear functions for $p(s_i)$ can be defined.

Tournament selection:
A third method of selection for variation is *tournament selection*. In each step, k individuals of the population are preselected randomly (independent on the fitness) and the best out of the k is chosen. k is a parameter of the method.

Several strategies exist for the **selection for survival**:

Generational replacement:
The simplest form is *generational replacement*, in which all parents are replaced by their offspring. This method has been used in traditional genetic algorithms in combination with fitness-proportionate selection for variation to enforce the selection pressure.

Steady state selection :
In *steady state selection* [293], the number of children produced by variation is smaller than the number of parents. Thus, a strategy is required to decide which parents are replaced. Several variants exist, such as *worst replacement*, and *oldest replacement* [132].

(μ, λ) **selection:**
In the (μ, λ)-ES, the μ parents are replaced by the best of the λ offspring ($\lambda \geq \mu$). The selection pressure can be increased by increasing the number of offspring μ.

$(\mu + \lambda)$ **selection:**
In the $(\mu + \lambda)$-ES, the best μ individuals are chosen from a temporary population containing the μ parents and the λ offspring.

Further methods exist that are used in combination with the selection strategies above, such as *elitism* (the best individuals always survive) [72], and *duplicate checking* (children identical to a parent are not included in the new generation) [86]. The latter is especially important in evolutionary algorithms with small population sizes.

3.2.4 The Evolutionary Variation Operators

Mutation and recombination operators depend on the coding of the candidate solutions of the optimization problem. In the following, operators for binary codings as used in GAs, operators for real-valued codings as used in ES, operators on finite state machines in EP, and operators on trees as used for example in genetic programming (GP) will be described to show their dependence on the underlying representation.

Genetic Operators on Bit Strings

The following operators on binary vectors are typically used in genetic algorithms. Let $A, B \in \{0,1\}^n$ be a bit-string (genome) representing a candidate solution.

One-point crossover:

This operator [142] works by cutting the two bit strings at a randomly selected cutting point p. The head of the first (second) is then connected to the tail of the second (first) chromosome. Thus, one-point crossover produces two solutions A' and B' with

$$A'_i = \begin{cases} A_i & \text{if} \quad i \leq p \\ B_i & \text{if} \quad i > p \end{cases} \qquad \text{and} \qquad B'_i = \begin{cases} B_i & \text{if} \quad i \leq p \\ A_i & \text{if} \quad i > p \end{cases} .$$

The following example illustrates the operation:

$$\begin{array}{l} A = 0110|100 \\ B = 1011|001 \end{array} \quad \longrightarrow \quad \begin{array}{l} A' = 0110|001 \\ B' = 1011|100 \end{array}$$

Two-point crossover:

In comparison to the one-point crossover, the two chromosomes are cut by the two point crossover [142] at two randomly chosen cutting points p_1 and p_2 ($p_1 \leq p_2$) resulting in three pieces. Thus, two solutions A' and B' are generated with

$$A'_i = \begin{cases} B_i & \text{if} \quad p_1 \leq i \leq p_2 \\ A_i & \text{otherwise} \end{cases} \qquad \text{and} \qquad B'_i = \begin{cases} A_i & \text{if} \quad p_1 \leq i \leq p_2 \\ B_i & \text{otherwise} \end{cases} .$$

The following example illustrates how the two-point crossover works:

$$\begin{array}{l} A = 011|01|00 \\ B = 101|10|01 \end{array} \quad \longrightarrow \quad \begin{array}{l} A' = 011|10|00 \\ B' = 101|01|01 \end{array}$$

A generalization if this crossover operator exists in which the bit strings are cut at k randomly chosen points, called a k-point crossover.

Uniform crossover:

The uniform crossover [3, 292] utilizes a crossover mask, to allow alternative forms of crossing-over. A crossover mask M is simply a bit string of the same length as the solution vector. The value of each bit M_i in the mask determines, for each corresponding gene in the child, from which parent it will receive the gene value:

$$A'_i = \begin{cases} B_i & \text{if} \quad M_i = 1 \\ A_i & \text{otherwise} \end{cases} \qquad \text{and} \qquad B'_i = \begin{cases} A_i & \text{if} \quad M_i = 1 \\ B_i & \text{otherwise} \end{cases}$$

Thus, one-point and two-point crossover are special cases of the uniform crossover: For the examples above, the corresponding masks are: $M = 0000111$ for one-point crossover, and $M = 0001100$ for two-point crossover. With uniform crossover, the 1-bits are uniformly distributed over the mask, typically occurring at each locus with a probability of 0.5. An example is provided in the following how uniform crossover works.

$$\begin{array}{l} A = 0110100 \\ B = 1011001 \end{array} \quad \rightarrow (M = 1011010) \rightarrow \quad \begin{array}{l} A' = 1111100 \\ B' = 0010001 \end{array}$$

Bit flip mutation:

Bit flip operators [142, 120] simply flip a small number of genes in the genome:

$$A = 01101\underline{1}001 \quad \rightarrow \quad A' = 01101\underline{0}001$$

Generally, there are two ways to implement such a mutation operator. The first way is to predefine a rate in which each bit in the genome is flipped. The other way is to predefine the number of bits to flip in the genome and to select the loci in the genome randomly. Assuming a bit string of length n, a mutation with a rate of $1/n$ per bit has almost the same effect as the mutation of a single randomly selected bit out of the n.

Inversion:

An alternative mutation operator is the inversion operator [142]: Two points are chosen along the length of the chromosome, the chromosome is cut at these points, and the substring is reversed.

$$A = 01|101100|1 \quad \rightarrow \quad A' = 01|001101|1$$

Mutation operators play only a secondary role in genetic algorithms. They are often used as "background operators" to add a source of diversity aimed to prevent a premature convergence. Mutation is typically applied to the offspring generated by crossover before the evaluation of the fitness.

Evolutionary Operators on Continuous Variables

Since evolution strategies mainly concentrate on continuous parameter optimization, the operators used in ES are described in the following as examples for operators on continuous search spaces.

Mutation:

Mutation can be realized by adding a random normally distributed number (with mean 0) to each component of a vector $x \in I\!R^n$. The resulting vector x' becomes:

$$x'_i = x_i + N(0, \sigma_i), \tag{3.3}$$

with $N(0, \sigma_i)$ denoting an independent normally distributed random number with expectation 0 and standard deviation σ_i [257, 269].

Recombination:

Recombination operators for continuous variables can be divided into discrete/intermediate and local/global operators [270, 139]. Let $a, b, c_1, \ldots, c_n, d_1, \ldots, d_n \in I\!R^n$ represent candidate solutions. The solution vector $a' = (a'_1, \ldots, a'_n)$ is generated as follows:

$$a'_i = \begin{cases} a_i \text{ or } b_i & \text{(discrete)} \\ \frac{1}{2}(a_i + b_i) & \text{(intermediate)} \\ c_{i,i} \text{ or } d_{i,i} & \text{(global, discrete)} \\ \frac{1}{2}(c_{i,i} + d_{i,i}) & \text{(global, intermediate)} \end{cases} \tag{3.4}$$

Self adaptation:

The self adaptation of the strategy parameter σ [269, 139] can be achieved by adding σ to the solution vector. Thus, the tuple (x, σ) is subject to variation. The mutated offspring (x', σ') is defined as:

$$\sigma'_i = \sigma_i \exp(\tau' \, N(0,1) + \tau \, N_i(0,1)), \tag{3.5}$$

$$x'_i = x_i + N(0, \sigma'_i). \tag{3.6}$$

where $i = 1, \ldots, n$ and the notation $N_i(\cdot, \cdot)$ indicates that the random variable is sampled anew for each value of i. τ and τ' are operator set parameters which define global and individual step sizes [16]. Variants of this self adaptation scheme have been proposed which are described, for example, in [140] in detail.

In contrast to GAs, in most applications of ES, mutation operators are used as the main search operators.

Evolutionary Operators for Finite State Machines

To evolve a system that is capable of showing intelligent behavior by predicting its environment and producing suitable responses, Fogel, Owens and Walsh [97] used finite state machines in their original evolutionary programming approach. In their model, the environment was described as a sequence of symbols taken from a finite alphabet. The problem was to find a finite state machine that would operate on the sequence of symbols thus far observed as to produce an output symbol to predict the next symbol to appear in the environment. In their algorithm, a finite state machine (FSM) consists of a finite number of states, for each of which there is an associated output symbol and next-state transition for every possible input symbol. For each FSM an initial state is defined. Figure 3.3 shows an FSM consisting of three states with an input alphabet of $\{0, 1\}$ and an output alphabet of $\{\alpha, \beta, \gamma\}$. In the EP approach, selection is based on a payoff that is defined for each output symbol dependent on the symbol to appear next in the environment.

Mutation:

Offspring machines are generated by five possible modes of random mutation based on the description of the finite state machine:

- change of an output symbol,
- change of a state transition,
- addition of a state,

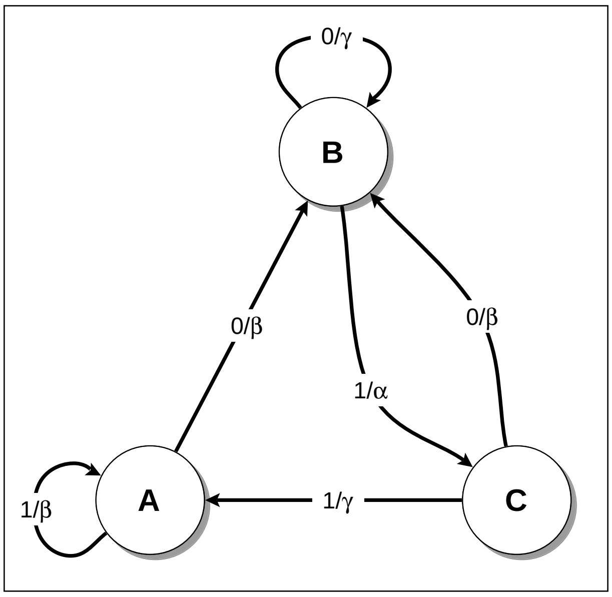


Figure 3.3: A Finite State Machine Consisting of three States

- deletion of a state, and
- change of the initial state.

No recombination has been used by Fogel *et al.* in the original EP approach.

Evolutionary Operators for Tree Data Structures

Koza [179] proposed the optimization of computer programs to solve problems. In his studies, he used the functional programming language LISP [206]. LISP functions can be interpreted as trees with arithmetical operators (functions) at the nodes and variables or constants (terminals) at the leafs. The following operators have been proposed for such trees.

Recombination:
Koza [179] proposed the exchange of subtrees between two selected parents. This preserves the correctness of the syntax and thus ensures that the offspring are feasible.

Mutation:
Due to the secondary role of mutation in GAs, mutation operators have not been used by most GP researchers. However, some unary operators exist [18]:

- The switching of siblings (if order matters),
- cycle operations,
- the growing of new subtrees,
- shrinking of a subtree to a leaf, and
- numerical terminal mutation.

3.2.5 The Relation between Genotype and Phenotype

In real world applications, the search space is defined by a set of objects, such as processing units, pumps or other technical devices, which have different parameters, e.g. energy consumption or capacities. These parameters are subject to optimization and thus constitute the phenotype space. In evolutionary algorithms, operators are often defined on abstract mathematical objects like binary strings. In these cases, a mapping between the genotype and the phenotype space is required to evaluate the fitness of a solution or to obtain the actual parameters for the optimization problem. In Figure 3.4, the relation between the genotype and the phenotype space is shown: While the genetic operators mutation and re-

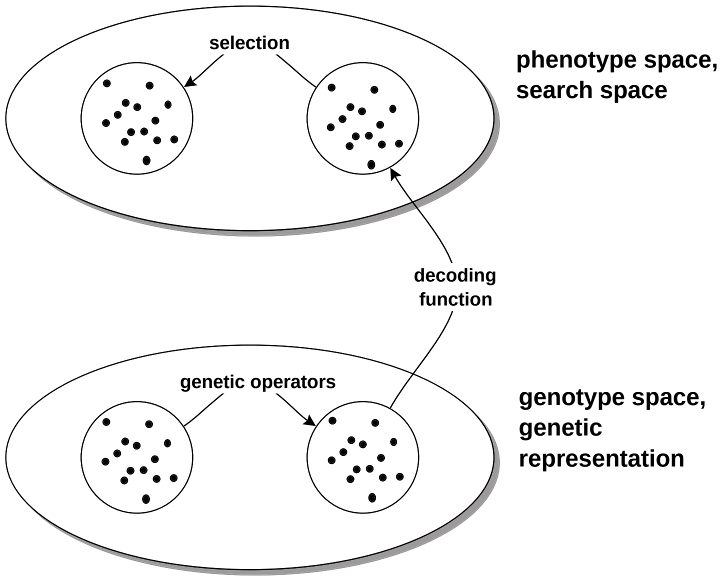


Figure 3.4: Phenotype and Genotype Space

combination operate on the elements in the genotype space, selection is performed within the phenotype space. A decoding function is required to map a genotype to its phenotype. Often, a representation as close as possible to the characteristics of the phenotype space is chosen, almost avoiding the need for a decoding function. This approach has the advantage that the introduction of additional nonlinearities or a higher computational complexity by a complex coding is avoided. On the other hand, organic evolution is based on the principle of using a genetic code. However, the decoding mechanism in nature is a highly complex and not well-understood process.

3.2.6 Application of EAs

Evolutionary algorithms are often referred to as black box optimization algorithms since they do not use any kind of domain knowledge for a given problem. The operators are defined independently of the problem: only the evaluation of the fitness function has to be implemented as long as the problem can be encoded as an unconstrained problem on bit vectors (GA) or as an unconstrained problem on continuous variables (ES). However, in many applications either implicit or explicit constraints are involved, which requires the definition of problem-dependent variation operators as in case of GP. Other examples are evolutionary programming on finite state machines or EAs for mixed integer programming problems.

In combinatorial optimization, the decision variables constituting a candidate solution are often discrete. Some problems have a solution space of binary vectors such as subset problems in which a solution is a subset of a larger set. These problems can be solved with a traditional GA, while others can not. Often, additional constraints – implicit or explicit – do not allow the application of a standard GA. For many problems, the solutions can be encoded with k-ary strings with or without implicit constraints. A common implicit constraint is that these k-ary strings must represent permutations of a set. EAs for these problems must contain specialized operators to ensure that crossover or mutation always produces feasible solutions. In the case of explicit constraints, other techniques are required for an EA to work, such as the use of penalty functions to penalize the fitness of infeasible solutions or repair algorithms that transform infeasible to feasible solutions.

Besides the presence of constraints, there are other characteristics that prevent the application of a standard EA such as multi-objective problems [99, 100], in which more than one fitness function has to be maximized, or dynamic problems [240] in which the fitness of a solution changes over time. These problems are current research topics in the field of evolutionary computation.

3.3 Other Biologically Motivated Optimization Techniques

Other methods inspired by nature have been proposed for solving combinatorial optimization problems besides evolutionary computation techniques. *Ant Colony Systems* (ACS) are inspired by the efficient cooperation of ants in ant colonies [76, 77, 75]. *Artificial Neural Networks* (ANN) [145, 174, 137, 208] are simple models of the central nervous system of the human brain. These approaches have been shown to be applicable to a wide range of combinatorial problems and can in some cases be extended to other domains of mixed parameter optimization.

3.3.1 The Ant Colony System

Real ants have developed an efficient way of finding the shortest path from a food source to their nest without using visual information. While searching for food, ants deposit pheromone on the ground and follow, in high probability, pheromone previously deposited by other ants. Assuming a single food source, more than one way to reach the source and initially equal probability for an ant to chose a path, more ants will visit the shortest path on average and therefore pheromone accumulates faster if they walk with approximately the same speed. If new ants arrive at a point where they have to decide on one or another path they prefer to choose the shorter path with higher probability. This in turn increases the pheromone on the shortest path such that after a while all ants will choose the shortest path.

The ant colony system is inspired by the behavior of real ants. In the ACS algorithm, a solution to a combinatorial optimization problem is constructed by agents (ants) which choose the values for the decision variables constituting a feasible solution stepwise. Each choice is – in analogy to real ants – a probabilistic choice proportionate to a global variable representing the amount of pheromone. Thus, in the ACS, the agents communicate indirectly via global, distributed memory: the vector or matrix of pheromone variables. After assigning a value to a component of a solution vector, a local pheromone update rule is applied. Furthermore, if an agent has finished in building a feasible solution, a global pheromone update rule is applied that takes the objective function value of the produced solution into account. This kind of learning imposed by the agents has some resemblance to *reinforcement learning* [162].

The Ant Colony System has been applied to various combinatorial optimization problems including the TSP [78], the QAP [288], and Vehicle Routing [110]. Since the beginning, the ACS has been developed further and its successor is called the *Ant Colony Optimization (ACO) meta-heuristic* [75]. ACO allows the combination of the ant system and a local search heuristic to improve its efficiency. This hybrid approach has lead to an improved performance for the TSP [78, 289], and for the QAP [111, 290].

3.3.2 Artificial Neural Networks

Artificial neural networks were mainly developed for the purpose of feature recognition or pattern classification and function approximation in, for example, time series prediction. However, it has been shown that the model can be modified for combinatorial optimization tasks. ANNs consist of neurons, the information processing units, and synapses, the links between the units, with associated weights. Common to most ANNs is a local updating rule that determines the state of a neuron dependent on the states of its other linked neurons and the weights of the links.

Classical ANNs are feed-forward networks, i.e., neurons are organized in layers and signals are processed from the input layer through one or more hidden layers to the output layer by applying a local update rule. Links are usually unidirectional and connect units from one layer with the next upper layer. ANNs for optimization based on the Hopfield model [145], however, are feed-back networks: the synapses are bidirectional and there is no layer structure.

The decision variables of a solution vector are derived from the states of the neurons. After the network is set up by choosing appropriate weights to describe the problem instance, an energy function depending on the states of the neurons and the weights is minimized. This is achieved by iteratively updating the neuron states with a local update rule based on appropriate *mean field equations* (MFT). The system then converges to a state that represents a feasible solution or a greedy repair heuristic is applied to obtain a feasible solution. Many variants of this scheme have been proposed for several combinatorial optimization problems, including the TSP [145, 253] and graph bipartitioning [249]. They differ in the way solutions are encoded and the types of MFT used. In some approaches, annealing schemes are used to prevent the system from getting stuck in poor local optima [248, 253]. The results obtained with these approaches are, however, of moderate quality. They are not comparable with other state-of-the-art heuristics.

Some other ANN techniques have been proposed based on deformable templates [248] or self-organizing maps [253]. These algorithms, such as *elastic nets* for the TSP [82], can be applied to geometric problems with low dimensionality. Besides their limited applicability they can not compete with other heuristics [158, 253].

3.3.3 Recent Developments

Recently, further approaches have been proposed for solving optimization problems. These approaches include (a) *immune system methods* [66], which are based on the principles of the information processing in the (human) immune system, (b) *particle swarm optimization* [170], which draws from the metaphore of human sociality, and (c) *cultural algorithms* [262], which model the evolution of cultural systems based upon principles of human social evolution. In contrast to memetic algorithms or other evolutionary algorithms, a cultural algorithm is a dual inheritance system with a population and a belief space. Thus, operators for both components and a communication protocol between the two spaces are required.

3.4 Greedy and Local Search Heuristics

Heuristics can be divided into construction heuristics and improvement heuristics. The former construct feasible solutions for a given optimization problem from scratch, while the latter take a feasible solution as input and try to find better solutions by stepwise transformations. Both types of heuristics can be implemented efficiently and are often capable of producing near optimum solutions for combinatorial optimization problems. There is a huge number of approaches for combinatorial optimization for both types. Construction heuristics include various types of highly problem-dependent heuristics. For example, for the TSP, construction heuristics are *nearest neighbor heuristics*, *insertion and addition heuristics*, the *greedy heuristic*, the *savings heuristic*, and heuristics based on spanning trees. Examples of improvement heuristics are *neighborhood search* based algorithms such as *local search*, *simulated annealing*, *threshold accepting* and *tabu search* which can be applied to various problems.

In the following, $f(s \in S)$ is referred to as an objective function of a maximization problem ($f(s) = m_P(I, s)$ if P is a maximization problem, and $f(s) = -m_P(I, s)$ otherwise).

3.4.1 Greedy Heuristics

Greedy algorithms are intuitive heuristics in which greedy choices are made to find solutions to a combinatorial optimization problem. Greedy heuristics are constructive heuristics since they construct feasible solutions for optimization problems from scratch by making the most favorable choice for a decision variable in each step. An appropriate measure for greedy choices is highly problem-dependent (sometimes even instance-dependent). Most important, a choice in each step depends on the decisions already made – the effects of future choices are unknown. Figure 3.5 shows the pseudo code for a problem that is encoded with a solution vector s of length n with $s = (s_1, \ldots, s_n) \in S = \{1, \ldots, k\}^n$ C denotes the candidate set of solution vector components for which no value as been chosen so far. The greedy choice of the pair (x, y) is performed in a way that the expected objective value $f(s)$ is maximized. This can be achieved, for example, by defining a partial function $f_p(s)$ that is defined over all s_i with $i \notin C$ that is maximized in each step. Greedy choices can be viewed as local decision rules and usually lead to sub-optimum solutions since the resulting solution at the end of construction is unknown and future decision may have a large impact on the resulting objective value of the solution.

Greedy algorithms have been proposed for various COPs such as Kruskal's polynomial-time algorithm for minimum spanning trees [183], the nearest neighbor and greedy heuristics

```
procedure Greedy($s \in S$): $S$;
  begin
    $C := \{1, 2, \ldots, n\}$;
    repeat
      Greedy choose a pair $(i, v) \in C \times \{1, \ldots, k\}$;
      $s_i := v$;
      $C := C \backslash \{i\}$;
    until $C = \emptyset$;
    return s;
  end;
```

Figure 3.5: The Greedy Algorithm

for the TSP [261], the min-max and differential greedy heuristics for graph bipartitioning [21], and and a greedy heuristic for binary quadratic programming [218].

3.4.2 Local Search Heuristics

Local search as well as *simulated annealing*, *threshold accepting*, and *tabu search* are *neighborhood search* based algorithms. Therefore, in the following a definition of neighborhoods for solutions of COPs is provided.

Neighborhoods

A neighborhood of a solution s to a combinatorial optimization problem is defined as a set of solutions which can be reached by applying an elementary operator $M : S \to \mathcal{P}(S)$ to the solution s. This set is denoted $\mathcal{N}(s)$:

$$\mathcal{N}(s) = M(s) \subset S \tag{3.7}$$

Such an operator is, for example, the *bit-flip* operator for binary vectors ($S = \{0, 1\}^n$). The *1-opt* or bit-flip neighborhood of x is defined as the set of all solutions obtained by flipping a single bit in x:

$$\mathcal{N}_{1\text{-}opt}(x)\mathcal{N}_{bit\text{-}flip}(x) = M_{bit\text{-}flip}(x) = \{x' \in S \mid d_H(x, x') = 1\}, \tag{3.8}$$

with d_H denoting the hamming distance of bit vectors. The elementary operator M defines a *move set* since the application of the operator yields a move m from a solution to another solution in its vicinity. The number of possible moves $|M|$, i.e., transformations of one solution into another, usually defines the neighborhood size.

Local search and the other variants of neighborhood search are based on the definition of a move set for a given COP and thus on a neighborhood defined by the move set.

Local Search

In the field of combinatorial optimization, local search algorithms (LSAs) have a long history since they are intuitive and very efficient. For example, the first local search algorithm for the traveling salesman problem was proposed in 1956/58 [59, 94], and a local search for the facilities location problem was developed before 1962 [13]. Figure 3.6 shows the general local search algorithm for a maximization problem: beginning with a feasible solution to the problem, a new solution with a higher objective f is searched in its neighborhood. If such a solution is found, the new solution is accepted and its neighborhood is searched for a better solution, and so on. The algorithm terminates when a local optimum is reached, i.e. when there is no solution in the neighborhood of the current best solution with a higher objective value.

```
procedure Local-Search($s \in S$): $S$;
  begin
    repeat
      Generate neighboring solution $s' \in \mathcal{N}(s)$;
      if $f(s') > f(s)$ then $s := s'$;
    until $\forall\, s' \in \mathcal{N}(s) :\ f(s') \leq f(s)$;
    return s;
  end;
```

Figure 3.6: Local Search

Local search (LS) is thus similar to simple *hill–climbing* with the difference that (a) the neighborhood of the current solution is searched systematically instead of randomly, and (b) the neighborhood search is repeated until a locally optimum solution is found. In some local search algorithms, a solution with the highest objective value is selected (best improvement) instead of the first improving solution found (first improvement).

The effectiveness of the heuristic depends highly on the choice of an appropriate neighborhood $\mathcal{N}$. The greater the neighborhood, the better the expected resulting objective may be, but enlarging the neighborhood quickly becomes impractical. For many combinatorial optimization problems, algorithms with neighborhoods of size greater than $O(n^2)$ with n denoting the problem size become inefficient and are therefore not used in practice.

One advantage of LS over other heuristics is that the configuration space can be searched very efficiently: instead of calculating the objective value of $s' \in \mathcal{N}(s)$, it is sufficient to calculate the difference $\Delta f = f(s) - f(s')$ (by utilizing problem–specific properties to avoid the explicit computation of $f(s)$ and $f(s')$) and to test whether Δf is less than zero. There are neighborhoods for almost every combinatorial optimization problem, where Δf can be computed much faster than $f(s')$. For example, in the traveling salesman problem, the calculation of Δf takes time $O(1)$, while the calculation of f takes $O(n)$. In other words, an LS algorithm is able to visit n solutions of the search space in the same time a traditional EA evaluates a single solution. LS shares this advantage with all other neighborhood–based search algorithms including the algorithms described below.

A disadvantage of LS is that the obtained solutions are – by definition – only local optima. Once a local optimum solution has been reached, the algorithm terminates. Therefore, to allow longer running times for finding better solutions, simulated annealing, tabu search and other neighborhood based search heuristics have been developed.

3.4.3 Simulated Annealing and Threshold Accepting

Simulated annealing [173] and *threshold accepting* [80] are variants of the simple local search introduced above. Both methods are neighborhood search methods, but they differ in the acceptance criterion for neighboring solutions. While in local search only better solutions are accepted in relation to their objective function value, simulated annealing and threshold accepting allow accepting solutions worse than the current solution.

Simulated Annealing

The origin of simulated annealing (SA) lies in the analogy of optimization and a physical annealing process [173]. In condensed matter physics, *annealing* is a thermal process for obtaining low-energy states of a solid in a heat bath. Roughly, the process can be described as follows. First, the temperature of the heat bath is increased to a maximum value at which the solid melts. Thus, all particles of the solid arrange themselves randomly. Afterwards, the temperature is carefully decreased until the particles of the melted solid reach in the ground state of the solid in which the particles are arranged in a highly structured lattice with minimum energy.

The physical annealing process can be simulated by computer programs using *Monte Carlo techniques* proposed by Metropolis et al. [220]. Given a current state i of the solid with energy E_i, a subsequent state j is generated by applying a perturbation mechanism, which transforms the current state into the next state by a small distortion, for instance by displacement of a single particle. If the energy difference $\Delta E = E_j - E_i$ is less or equal to zero, the state j is accepted as the current state. If the energy difference is greater than zero, the state j is accepted with probability $\exp(-\Delta E/(k\,T))$, where T denotes the temperature of the heat bath and k the *Boltzmann constant*. The acceptance rule described above is known as the *Metropolis criterion*.

In simulated annealing, the Metropolis criterion is used to generate sequences of solutions of combinatorial optimization problems. Thus, the solutions of a COP can be interpreted as the states of the physical system, and the objective value of a solution can be regarded as the energy of the state (for maximization problems: $E = -f$). The general outline of SA is illustrated in Figure 3.7. To successfully apply SA to a COP, a *cooling schedule* $T : I\!N \rightarrow I\!R$ must be determined, which provides an initial value for the temperature t as well as an updating rule $T(\cdot)$ for the temperature. Various cooling schedules have been proposed including static and dynamic schedules. An overview is presented in [2]. However, there is no accepted methodology for choosing the schedule, since finding the optimal schedule is itself an optimization problem.

Threshold Accepting

Threshold accepting (TA) has been proposed as an alternative to simulated annealing [80]. Here, the Metropolis criterion is replaced by a much simpler (computationally inexpensive) acceptance criterion. A neighboring solution is accepted, if the difference of the objective values of the current and the neighboring solution is below a threshold θ. Thus, in TA the acceptance criterion is deterministic as opposed to the one used in SA. In Figure 3.8, the pseudo code of TA is given. Similar to simulated annealing, initial value and update rule for θ have to be provided in form of the function Θ [80].

```
procedure Simulated-Annealing($s \in S$) : $S$;
  begin
    $t := T(0)$, $n := 1$;
    $s_{best} := s$;
    repeat
      Generate neighboring solution $s' \in \mathcal{N}(s)$;
      $\Delta f := f(s) - f(s')$;
      if $\Delta f \leq 0$ or $\exp(-\Delta f/t) > random[0, 1)$ then $s := s'$ ;
      if $f(s) > f(s_{best})$ then $s_{best} := s$;
      $t := T(n)$;
      $n := n + 1$;
    until termination criterion fulfilled;
    return $s_{best}$;
  end;
```

Figure 3.7: Simulated Annealing

```
procedure Threshold-Accepting($s \in S$) : $S$;
  begin
    $\theta := \Theta(0)$, $n := 1$;
    $s_{best} := s$;
    repeat
      Generate neighboring solution $s' \in \mathcal{N}(s)$;
      $\Delta f := f(s) - f(s')$;
      if $\Delta f < \theta$ then $s := s'$;
      if $f(s) > f(s_{best})$ then $s_{best} := s$;
      $\theta := \Theta(n)$;
      $n := n + 1$;
    until termination criterion fulfilled;
    return $s_{best}$;
  end;
```

Figure 3.8: Threshold Accepting

3.4.4 Tabu Search

An essential feature of tabu search (TS) [117] is the use of memory. As with SA and TA, the acceptance criterion of neighboring solutions allows the selection of solutions with smaller objective values. The acceptance criterion is deterministic since it always chooses the neighbor with highest fitness. To prevent the search from getting stuck in endless loops, a memory of the search process is utilized. A solution that has been recently visited is included in a *tabu list* and therefore will not be considered as a candidate for the next solution to visit. An outline of this method is provided in Figure 3.9. However, the maintenance of the tabu list

```
procedure Tabu-Search($s \in S$) : $S$;
  begin
    $T := \emptyset$;
    $s_{best} := s$;
    repeat
      Find best solution $s' \in \mathcal{N}(s)$ with $s' \notin T$;
      $s := s'$;
      $T := T \cup s$;
      if $f(s) > f(s_{best})$ then $s_{best} := s$;
    until termination criterion fulfilled;
    return $s_{best}$;
  end;
```

Figure 3.9: Tabu Search

and the searching within the list is often too time consuming to be practical. An alternative is not to store solutions in the tabu list but the move that has lead to the solution. To be more precise, for each possible move, a flag in a data structure indicates whether the move is forbidden or not. Since this procedure is in some cases too restrictive, *aspiration level conditions* have been incorporated. The aspiration level of a move indicates if it should be considered in the search despite of its tabu status. For example, if the move leads to a solution better than the previous best one, the move should be allowed regardless whether it is tabu or not. The neighborhood selection in Figure 3.9 has thus to be modified to:

```
Find best move $m$ for $M(s)$ with $(s' = m(s) \in \mathcal{N}(s)$ and $(m \notin T_m$ or $a(m) \geq A(m))$;
```

$M(s)$ denotes the move set for s, T_m denotes the tabu list of moves, $a(m)$ the aspiration level of move m, and $A(m)$ the aspiration threshold for m.

Alternatively, more than one tabu list and more than one aspiration function can be used.

There are various extensions to this basic scheme: for example, tabu lists of variable size can be used to improve the efficiency of the algorithm [294]. Furthermore, the intensification of the search as described by the procedure above can be combined with a diversification phase. If the TS was unable to find a new best solution for a predefined number of iterations, the search may be concentrated on another region of the solution space. The job of the diversification phase is to determine promising alternative regions based on the memory of the search. An effective algorithm using diversification has been published under the name *reactive tabu search* [23].

3.5 Towards Memetic Algorithms

It has been argued that it is essential to incorporate some form of domain knowledge into evolutionary algorithms to arrive at highly effective search [125, 106, 15, 221]. There are many ways to achieve this, for example by hybridization: by combining evolutionary algorithms with other problem-dependent heuristics, the efficiency of EAs has been increased in many cases. In the following, the combination of EAs with local search will be addressed since this symbiosis has been shown to be very promising. The basic idea of neighborhood search (NS) itself is problem independent. However, several local search algorithms have been proposed that utilize some form of domain knowledge to be more efficient and effective.

Hybrid EAs combined with neighborhood search can be divided into algorithms exploiting the *Baldwin effect* and algorithms based on Lamarckian evolution [310, 108]. The former use neighborhood search to modify the landscape/structure of the problem: before the fitness is evaluated, a neighborhood search is applied. The changes (improvements) made by the neighborhood search are not saved in the individual, thus learned traits during lifetime of the parents are not inherited by their offspring. This resembles natural evolution in which an individual's acquired traits during life-time have no impact on the genetic code that gives rise to a new organism. Opposed to this form of evolution is Lamarckian evolution, in which acquired traits of an organism influence the genetic code of the organism. This form of evolution has its origin in the evolution theory of Lamarck (1809) which has been shown to be wrong with the publication of Darwin's work.

However, it has been shown that the Lamarckian approach has its advantages in optimization: in most hybrid algorithms, the individuals are altered by the variation operators and the neighborhood search.

3.5.1 Combination of Evolutionary Algorithms and Neighborhood Search

Local search is a very efficient search technique in combinatorial optimization since problem characteristics can be utilized to speed up the neighborhood search process and thus more solutions per time interval can be visited. This is, for example, realized by finding an efficient way to calculate Δf as mentioned above. The rigorous selection of solutions by the acceptance criterion allows for a fast "convergence" towards high quality solutions. This form of search is referred to as *intensification*, since many solutions are visited in a relatively small region of the search space and the best solution found is returned. If the local optimum solutions with respect to a straight-forward local search are not satisfying, advanced techniques to escape from local optima can be used as in simulated annealing or tabu search.

The disadvantages of local search are obvious. Neighborhood search based algorithms are improvement heuristics and thus highly depend on the starting solution. The resulting solution is only locally optimal even in SA and TS due to the chosen annealing schedule in SA or the limited number of iterations in TS. The search is only guided by local information, no other information is utilized (advanced tabu search techniques are an exception). The choice of the starting solution in a neighborhood search is extremely critical: an inappropriate choice may lead to a local optimum with low objective value and thus high distance to the optimum with respect to the objective function. In population-based algorithms such as EAs, the probability of choosing an unfavorable start configuration is minimized due to the selection of a large number of starting solutions distributed over the whole search space. The search is then focused on promising regions of the search space by successively narrowing the region(s) until the search is said to have converged.

The consequent combination of neighborhood search and evolutionary algorithms requires that the components of the EA are seen in a different light. Since in a Lamarckian approach, all individuals represent local optimum solutions, the role of the variation operators shifts: The operators are no longer required to to find solutions with higher average fitness. They just have to produce solutions in a region of the search space that lies in the basin of attraction of a local optimum with higher fitness. Therefore, it is desired that the local search applied after variation does not end in a previously discovered solution. Instead, a variation operator should be innovative in the sense that it is capable of using knowledge about the search state to find new promising attractor regions of local optima with higher fitness. A hybrid evolutionary approach with variation operators fulfilling these requirements has not much in common with the random process of the crossing-over of chromosomes or the erroneous copying of genes as observed in nature.

3.5.2 Cultural Evolution

Genetic evolution is not the only form of evolution as pointed out by the biologist Richard Dawkins [71]. *Cultural evolution* is orders of magnitude faster than genetic evolution. In analogy to genetic transmission, cultural transmission is the flow of information in an evolutionary process. Genes can be thought of as parts of a chromosome with sufficient copying-fidelity to serve as a viable unit of natural selection. Their counterparts in cultural evolution are called *memes* as an abbreviation of the Greek word *mimeme* [71]. A meme, as a unit of cultural transmission, is replicated by *imitation*. Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or the building of arches. Like in genetics, these replicating units are subject to selection, since ideas, for example, have opposites they compete with and the human brain is only capable of holding a limited number of ideas. During transmission, variation occurs - often to a high extent. For example, ideas for building a scientific theory are often modified, replaced or put into another perspective.

There are some important differences between genetic and cultural evolution. First, cultural evolution is a much faster and less resource intensive process. Second, in cultural evolution, variation is seldom a product of copying errors or the random exchange of co-adapted units of information. In science, the raw combination of ideas does not always lead to an improved theory – in fact, this is rarely the case. A scientist usually passes on an idea after blending it with his own and verifying its viability. The process of recombining ideas allows for innovations, since knowledge or ideas from other fields and thus new memes can be integrated. Biological evolution does not incorporate an innovative component that allows experimenting with alleles not present in the parental chromosomes except by pure chance. The probability of mutating the “right” genes of an organism with high fitness diminishes rapidly with increasing fitness. Finally, natural evolution is an open-ended process, while cultural evolution is a goal-oriented process in which the information transmitting agents act consciously.

3.5.3 The Memetic Algorithm

Algorithms with closer analogy to cultural evolution than to biological evolution are called *memetic algorithms* [226, 230]. The new terminology accounts for the fact that the search strategy behind these algorithms differs significantly from other evolutionary algorithms. There are several design goals important for effective evolutionary search, which are fulfilled by MAs:

- Optimization algorithms should be efficient, i.e., they should be capable of producing acceptable solutions in short time.
- Optimization algorithms are by definition goal oriented. Viewing the individuals representing candidate solutions as agents, the individuals can be regarded as conscious entities in the search process subject to cooperation and competition.
- Due to the fact that the massively parallel, blind genetic search is resource intensive, only small populations can be evolved in short time. Thus, the inadequate sampling of the search space leads to a loss of diversity/genetic information and thus to a fast premature convergence. To overcome this drawback, a diversification mechanism for introducing 'innovative ideas' (new solution components) is required. This can be achieved to some degree by neighborhood search. However, alternative recombination schemes are often required to introduce diversification and thus innovation for which no biological analogy exists, e.g. operators that simulate behaviors of rebellious agents (rebel/obsequent versus conciliator) [30].

The memetic algorithms introduced in this work are based on the template shown in Figure 3.10. In this scheme, all individuals in the population represent local optima, which is ensured by applying local search after generation and after application of the evolutionary variation operators. Unlike in GAs, recombination and mutation are applied independently of each other. Greedy construction heuristics can be used for the generation of the solutions during initialization of the population and in the variation operators. Furthermore, a diversification scheme is included, which is borrowed from the CHC algorithm [88]: if the population is converged, all individuals in the population are mutated except the best one, and a local search is applied afterwards to ensure that all individuals are local optima. Thus, the diversification is a high level mutation operator acting on populations in comparison to the evolutionary variation operators acting on individuals. Thus, this form of mutation can be regarded as 'meta mutation'. This mechanism is required, since only small populations can be used due to the relatively long computation times of local search. This in turn causes a rapid convergence to sub-optimum regions in the search space which can be overcome with this restart technique.

Selection for reproduction is performed in a purely random fashion while selection for survival is performed as in the $(\mu + \lambda)$-Evolution Strategy: the locally optimized offspring generated by recombination and mutation together with the current population form a temporary population from which the best individuals are selected. However, each individual is selected only once and identical copies are removed.

The MA can be described by the tuple (p, c_r, c_m) in which p denotes the number of individuals in the population, c_r the recombination application rate, and c_m the mutation application rate. The number of offspring generated by recombination is thus $p \cdot c_r$, and the number of offspring generated through mutation is $p \cdot c_m$. In the ES notation, the algorithm can be described by a $(\mu + \lambda)$ strategy with $\mu = p$, and $\lambda = p \cdot (c_r + c_m)$.

Other variants of memetic algorithms [227, 121] differ from the basic MA framework above in that they are based on asynchronous models developed for transputers or message-passing systems. The framework introduced here is sometimes referred to as genetic local search [1, 301, 105, 175, 314]. Other researchers simply call their memetic approaches hybrid evolutionary algorithms without explicitly distinguishing them from other methods of hybridization [121, 93, 41, 44, 184]. Some of the published memetic approaches employ tabu search [93] or simulated annealing [41, 227] instead of local search.

```
procedure MA;

  begin
    for $j := 1$ to $popsize$ do
      $i :=$ generateSolution();
      $i :=$ Local-Search($i$);
      add individual $i$ to $P$;
    endfor;
    repeat
      for $i := 1$ to $\#recombinations$ do
        select two parents $i_a, i_b \in P$ randomly;
        $i_c :=$ Recombine($i_a, i_b$);
        $i_c :=$ Local-Search($i_c$);
        add individual $i_c$ to $P$;
      endfor;
      for $i := 1$ to $\#mutations$ do
        select an individual $i \in P$ randomly;
        $i_m :=$ Mutate($i$);
        $i_m :=$ Local-Search($i_m$);
        add individual $i_m$ to $P$;
      endfor;
      $P :=$ select($P$);
      if $P$ converged then $P :=$ mutateAndLS($P$);
    until terminate=true;
  end;
```

Figure 3.10: The Memetic Algorithm

3.5.4 Special Cases of the Memetic Approach

The simplest method of improving a single LS is to start it multiple times with different starting solutions and record the best solution found. This approach, called *multi–start local search*, can be realized by using randomly generated solutions as starting points, or using special randomized heuristics for producing the starting solutions.

Another, more advanced technique is called *iterated local search* (ILS): here, a starting solution is generated (randomly) and a LS algorithm is applied to turn the solution into a local optimum. Then, a new starting solution is generated by mutating the current solution (best local optimum found). A local optimum is then computed from this solution by LS. If the new local optimum has a higher fitness, it is accepted as the new current solution from which new starting solutions are generated. These steps are repeated until a user–defined termination criterion is fulfilled. The pseudo code is given in Figure 3.11. The ILS algorithm has first been proposed by Baum [24] under the name *iterated descent*. Johnson also used this technique for solving the TSP: the *iterated Lin-Kernighan algorithm* (ILK)[156, 158] is based on the Lin-Kernighan LS heuristic [192] and is known to be one of the best heuristics for the TSP today.

```
procedure Iterated-Local-Search : $S$;
  begin
    Create starting solution $s$;
    $s$ := Local-Search($I$, $s$);
    repeat
      $s'$ := Mutate($I$, $s$);
      $s'$ := Local-Search($I$, $s'$);
      if $f(I, s') > f(I, s)$ then $s := s'$;
    until terminate = true;
    return $s$;
  end;
```

Figure 3.11: Iterated Local Search

The meta–heuristic used in ILS is a simple hill–climbing algorithm operating on local optima: the current solution is modified randomly and only better local optima are accepted. To allow the algorithm to accept local optima with worse fitness, the hill–climber in ILS can be replaced by simulated annealing. In this algorithm, called *large step markov chain algorithm* (LSMC) [202], a newly generated local optimum is accepted according to the Metropolis criterion depending on the current temperature [203]. Again, the algorithm operates only on a single solution.

Although these meta-heuristics are not refered to as memetic algorithms since they are no population-based approaches, a memetic algorithm can act as a iterated local search or a LSMC when the population size is set to one and the recombination application rate is set to zero. The population-based memetic algorithm has been proven to be more flexible and more robust than single-agent ramdomized local search. The exploration of the search space is ensured in an algorithm starting its searches from multiple points in the search space, while single-solution approach suffers from a high dependence on the starting configuration.

3.6 Summary

In this chapter, an introduction to evolutionary computation has been given. Evolutionary algorithms have been introduced which are inspired by the process of natural evolution. It has been shown how these evolutionary algorithms such as genetic algorithms, evolution strategies, and evolutionary programming work and how they can be applied to new problems: the choice of an adequate problem representation, the definition of the fitness function, and the realization of one or more variation operators (recombination and mutation) in case of a non standard representation are necessary for the new evolutionary algorithm. In respect to most practical applications including combinatorial optimization problems, it has been argued that the incorporation of domain knowledge into evolutionary algorithms is crucial for achieving a performance comparable to other domain-specific optimization approaches.

Two other naturally inspired optimization techniques for solving combinatorial problems have been briefly described: *ant colonies* and *artificial neural networks*. For these algorithms the incorporation of domain knowledge is also required to enhance their performance.

Greedy and *local search algorithms* have been described as examples of highly efficient search algorithms which can exploit problem characteristics by incorporation of knowledge about the problem. However, these algorithms have some limitations. Greedy choices within a greedy construction algorithm are only locally optimal in a sense that decisions are made about the objective value of a solution without actually knowing the feasible solution resulting at the end of construction. Thus, choices in the beginning of the construction may turn out to be wrong in later steps. Local search heuristics, on the other hand, have the disadvantage that the solutions are per definition only locally optimum solutions. The resulting solution may highly depend on the starting configuration, and usually only local information is used during the search. However, these heuristics are very well suited for incorporation into evolutionary algorithms. The resulting algorithms, called *memetic algorithms*, have been shown to have strong resemblance with cultural evolution. Moreover, they are highly effective: memetic algorithms are among the best published approaches for various combinatorial optimization problems, as will be shown in the following chapters. Other meta heuristics, such as *large step markov chains* and *iterated local search*, can be regarded as special cases of the memetic approach.

In the following chapter, the analysis of fitness landscapes for combinatorial optimization problems is discussed in order to enable performance prediction and to provide a guideline for the development of memetic algorithms.

Chapter 4

Performance Prediction and Fitness Landscape Analysis

4.1 Introduction

The number of publications is growing rapidly in the fields of heuristic optimization and evolutionary computation. However, most of the work published deals with an existing algorithm applied to a new problem or a new algorithm applied to well-known (test) problems. Often, it is simply shown *that* a given algorithm works (for the problem instances tested) but not *why*. In most cases, it would be more valuable to know for which types of problems (or problem instances) an algorithm performs well or which parameters of an algorithm are best under given circumstances. Being capable of predicting the performance of an algorithm allows deciding which parameters are best for an algorithm or designing alternative algorithms that perform better than the ones at hand. In evolutionary computation, much progress has been made by theoretical investigations of evolution strategies or other evolutionary algorithms to understand how and when these techniques work. However, the mathematical analysis is usually limited to algorithms based on relatively simple models. For complex techniques like hybrid algorithms that incorporate as much domain knowledge as possible, the theoretical analysis appears to be too complicated. Fortunately, there are other ways to analyze such algorithms, i.e., by controlled experiments.

This chapter is concerned with the experimental analysis of memetic algorithms for combinatorial optimization problems. Such an analysis has to be performed ideally without actually conducting experiments with a heuristic optimization algorithm to be suitable for performance prediction. Thus, the structure of the search space is investigated in a first step, so that in a second step the characteristics of the search space can be used to find a search strategy that is likely to perform well.

The structure of the search space can be analyzed by utilizing the notion of the fitness landscape: the search space is regarded as a spatial structure were each point (solution) has a height that constitutes forming a landscape surface. In this chapter, proposed statistical measures for fitness landscapes are described and it is shown how they can be used to predict the performance of certain types of memetic algorithms. In particular, an autocorrelation or a random-walk correlation analysis is proposed to find the correlation length of a landscape and thus a measure of its ruggedness. A measure of the global structure of the landscape in terms of the distribution of local optima is shown to be the fitness distance correlation coefficient which can be obtained by a fitness distance correlation analysis. Some other aspects of performance evaluation and relative algorithm performance are also considered.

Parts of this chapter have been published in [214].

4.2 Performance Evaluation of Heuristics

The design of experiments is an important topic in experimental studies of heuristic algorithms. It has been argued in [144] that competitive testing as conducted by most researchers has several pitfalls.

The most obvious difficulty is to make a competition of heuristics fair. Differences between machines including CPU performance, memory configuration, operating system, and machine loads do not allow to compare running times directly. They can be overcome by using a single machine to perform all experiments with the algorithms in the comparison if the programs or source codes of all algorithms are available to the investigator. But there are other aspects that have to be considered for a fair comparison such as coding skill, tuning and effort invested. Not all authors use the same coding techniques due to different levels of experience. Furthermore, it is not even clear which technique is best for a given algorithm. Often, some of the investigators have tuned the parameters of their algorithm for a given set of problems while others have not. Another aspect is the effort invested in the development of an algorithm. Different data structures might, for example, be implemented for efficiently performing a time consuming operation. After weeks or even months of development and testing, the best suited data structure is selected, often resulting in a much improved performance. For example, the Lin-Kernighan heuristic for the TSP [192] is a complex algorithm for which many implementations with large performance differences in both solution quality and computation time exist [158, 210, 261, 44, 235, 165]. The order in which moves/submoves are considered, the number of nearest neighbors per node, the levels of backtracking, and the used data structure for performing moves are examples of implementation issues that are important for the efficiency of the heuristic. Three alternative data structures for performing moves have been proposed that show considerable performance differences to a straightforward implementation [104]. The code used in this work has been developed in one man year but is still inferior to the implementation of Johnson *et al.* [158].

Another important issue in experimental testing is the choice of problem instances. Randomly generated problem instances generally do not resemble real problems. On the other hand, using benchmark problems is also dangerous. Benchmark problems are collected after they have appeared in publications that report the performance of a new approach. These publications would not have appeared unless the new algorithm performed well on most of the instances introduced. Thus, instances that are solved easily by a new heuristic have a selective advantage. These instances may begin to design algorithms once they are accepted since new algorithms are published only if they perform well on the established benchmark set. Hence, the problem of finding an adequate, representative problem set is unsolved. It even appears to be impossible to recognize a representative set as representative!

Hooker [144] argues that competitive testing diverts time and energy from more productive experimentation. Since competitive testing only tells which algorithm is better than another on a set of problems but not why, he proposes to follow a more scientific approach for evaluating heuristics. Hooker argues that with experimental design methods such as factorial design insights can be gained which factors have a significant impact on the performance of a heuristic and which not.

However, such an approach also has some disadvantages. First, such experimentation is very time consuming and therefore not always practical. Second, if the hypothesis to be tested is wrong, all conclusions drawn from the results of the experiments are useless. Furthermore, if the factors selected or the test instances used are not representative, the conclusions drawn have no general meaning. Finding the appropriate factors and adequate test instances itself requires experiments.

4.3 The No Free Lunch Theorems

The *no free lunch theorems for optimization* [312] discuss the performance of black box optimization algorithms such as genetic algorithms, simulated annealing and hill–climbing in relation to the problems they are solving. In particular, it has been shown that all black-box optimization techniques have the same average behavior over all problems $f : X \to Y$. Thus, if an algorithm is effective on average for one class of problems, then it must do worse over the remaining problems. In fact, if the algorithm outperforms random search for a particular class of problems, it follows that it is worse than random search on the remaining problems.

These theorems support the claim that comparisons between such techniques are useless and that there is no point in believing that EAs are superior to other optimization techniques. However, it has been argued that concentrating on a subset of all problems $F' \subset F = \{f : X \to Y\}$ (restricted black box optimization) in fact leads to performance differences of the optimization techniques [79].

As discussed in the last chapter, in practical applications, as much domain knowledge as available is used to develop hybrid optimization algorithms that are capable of exploiting the structure of the problem to solve. Undoubtedly, these highly specialized algorithms perform better than other more general methods on the problems they have been developed for. On the other hand, they may not perform as good as a general method on other problems or they are even not applicable at all.

The theorems stress even more the importance of finding exploitable structural properties in optimization problems and thus imply a demand for adequate analysis techniques. Furthermore, they indicate the danger of defining problem instances that do not reflect the characteristics of real world instances and thus favor optimization techniques that may be inferior on real world instances than others.

4.4 Fitness Landscapes

The notion of *fitness landscapes* [313] has been proven to be a very powerful concept in evolutionary theory. Moreover, the concept has been shown to be useful for understanding the behavior of combinatorial optimization algorithms and can help in predicting their performance. Viewing the search space, i.e. the set of all (candidate) solutions, as a landscape, a heuristic algorithm can be thought of as navigating through it in order to find the highest peak of the landscape; the height of a point in the search space reflects the fitness (objective) of the solution associated with that point.

More formally, a fitness landscape $\mathcal{L} = (X, f, d)$ of a problem instance for a given combinatorial optimization problem consists of a set of points (solutions) X, an objective function

$f : X \rightarrow I\!R$, which assigns a real–valued fitness to each of the points in X, and a distance measure d, which defines the spatial structure of the landscape. The fitness landscape can thus be interpreted as a graph $G_{\mathcal{L}} = (V, E)$ with vertex set $V = X$ and edge set $E = \{(x, y) \in X \times X \mid d(x, y) = d_{min}\}$, with d_{min} denoting the minimum distance between two points in the search space. The diameter $diam\, G_{\mathcal{L}}$ of the landscape is another important property; it is defined as the maximum distance between the points in the search space.

Alternatively, a landscape can be defined as the tuple $\mathcal{L} = (X, f, \mathcal{N})$, where the neighborhood $\mathcal{N}$ of a solution defines the edges in the graph $G_{\mathcal{L}}$ ($E = \{(x, y) \in X \times X \mid y = \mathcal{N}(x)\}$). Since distance and neighborhood are closely related, one can be expressed with the other. For example, a neighborhood can be defined as $\mathcal{N}_k(x) = \{y \in X : d(x, y) \leq k\}$. On the other hand, a distance metric can be defined via the (minimum) number of applications of an elementary operator m to transform one solution into another.

For binary coded problems ($X = \{0, 1\}^n$), the graph $G_{\mathcal{L}}$ is a hypercube of dimension n, and the distance measure is the Hamming distance between bit strings. The minimum distance d_{min} is 1 (one bit with a different value), and the maximum distance is $diam G_{\mathcal{L}} = n$. The elementary operator defining the Hamming distance is the bit-flip operator that flips a single bit in the solution vector.

In the context of combinatorial optimization, the fitness landscape is often called *cost surface*, since a cost has to be minimized in many COPs instead of a fitness maximized. In the following, we refer to f as the objective function regardless of dealing with a minimization or maximization problem and we assume a maximization problem if we talk about fitness values.

4.4.1 Properties of Fitness Landscapes

Several properties of fitness landscapes are known to have influence on the performance of heuristic optimization algorithms. The following characteristics are important:

- The distribution (mean and variance) of the objective function,
- the landscape ruggedness,
- the number of local optima (peaks) in the landscape,
- the distribution of the peaks in the search space,
- the number of iterations needed to reach a local optimum,
- the structure of the basins of attraction of the local optima, and
- the presence and structure of neutral networks.

Statistical methods have been proposed to measure some of the properties while measures for others can not be determined easily. Furthermore, some of the properties are strongly related in many landscapes.

In the following, proposed measures for properties of fitness landscapes are discussed.

4.4.2 Preliminaries

Before measures of landscape properties can be discussed, some statistical terminology has to be introduced: the mean, variance, standard deviation and covariance of measurement variables.

The mean $\overline{X}$ and variance $\sigma^2(X)$ of a measurement variable $X = X_1, \ldots, X_n$ is defined as

$$\overline{X} = \langle X \rangle = \frac{1}{n} \sum_{i=1}^{n} X_i \tag{4.1}$$

and

$$\sigma^2(X) = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \overline{X})^2. \tag{4.2}$$

The square root of the variance – the standard deviation – is denoted by $\sigma(X)$.

The covariance of two variables $X = X_1, \ldots, X_n$ and $Y = Y_1, \ldots, Y_n$ is defined as

$$Cov(X, Y) = \langle XY \rangle - \langle X \rangle \langle Y \rangle = \frac{1}{n} \sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y}). \tag{4.3}$$

Correlation coefficients are usually defined as the quotient of covariance of two measurement variables and the product of the standard deviations of the two variables.

The mean objective value and the variance (or standard deviation) provide simple measures of a landscape. The mean objective value $\overline{f}$ of a landscape $\mathcal{L} = (X, f, d)$ with $n = |X|$ candidate solutions is defined as

$$\overline{f} = \langle f \rangle = \frac{1}{|X|} \sum_{x \in X} f(x) \tag{4.4}$$

and can be easily obtained in experiments. The fitness variance σ_f^2 is defined as

$$\sigma_f^2 = \sigma^2(f) = \frac{1}{|X|} \sum_{x \in X} (f(x) - \overline{f})^2 = \frac{1}{|X|} \sum_{x \in X} f^2(x) - \overline{f}^2 = \langle f^2 \rangle - \langle f \rangle^2. \tag{4.5}$$

Some other statistical properties depend on the mean and variance of the landscape as for example the fitness distance correlation and the autocorrelation functions described below.

4.4.3 Fitness Distance Correlation

An important measure is the *fitness distance correlation (FDC) coefficient*, proposed in [160] as a measure for problem difficulty for genetic algorithms. The FDC coefficient ϱ is defined as

$$\varrho(f, d_{opt}) = \frac{Cov(f, d_{opt})}{\sigma(f)\,\sigma(d_{opt})} = \frac{\langle f d_{opt} \rangle - \langle f \rangle \langle d_{opt} \rangle}{\sqrt{(\langle f^2 \rangle - \langle f \rangle^2)(\langle d_{opt}^2 \rangle - \langle d_{opt} \rangle^2)}} \tag{4.6}$$

and determines how closely objective and distance to the nearest optimum in the search space are related. The FDC coefficient can be estimated by

$$\varrho(f, d) \approx \frac{1}{\sigma(f)\sigma(d)} \frac{1}{m} \sum_{i=1}^{m} (f_i - \overline{f})(d_i - \overline{d}) \tag{4.7}$$

given a set of points $x_1, x_2, \ldots, x_m$ with $f_i = f(x_i)$ denoting the objective value and $d_i = d_{opt}(x_i)$ denoting the shortest distance to a global optimum solution. Note that there may be more than one global optimum. If fitness increases when the distance to the optimum becomes smaller, then search is expected to be easy for selection–based algorithms, since there is a "path" to the optimum via solutions with increasing fitness. A value of $\varrho = -1.0$ ($\varrho = 1.0$) for a maximization (minimization) problem indicates that fitness and distance to the optimum are perfectly related and that search promises to be easy. A value of $\varrho = 1.0$ ($\varrho = -1.0$) means that with increasing fitness the distance to the optimum increases, too.

4.4.4 Autocorrelation

According to Kauffman [169], a fitness landscape is said to be *rugged* if there is low correlation between neighboring points of the landscape, and a landscape is *smooth* if there is high correlation between neighboring points. The ruggedness of a landscape has high influence on the performance of search heuristics: generally, the more rugged a landscape, the harder the problem instance for heuristic approaches.

The Autocorrelation Function

To measure of the ruggedness of a fitness landscape, Weinberger [306] suggests the use of (auto)correlation functions. The *autocorrelation function* $\rho(d)$ [277, 306] reflects the correlation of points at distance d in the search space. Let $X^2(d)$ be the set of all pairs of points in the search spaces with distance d:

$$X^2(d) = \{(x, y) \in X \times X \mid d(x, y) = d\}, \tag{4.8}$$

and let $|X^2(d)|$ denote the number of pairs in the set $X^2(d)$. Then $\rho(d)$ is defined as

$$\rho(d) = \frac{\langle f(x)f(y)\rangle_{d(x,y)=d} - \langle f\rangle^2}{\langle f^2\rangle - \langle f\rangle^2} = 1 - \frac{\langle (f(x) - f(y))^2\rangle_{d(x,y)=d}}{2(\langle f^2\rangle - \langle f\rangle^2)} \tag{4.9}$$

where $\langle f(x)f(y)\rangle_{d(x,y)=d}$ denotes the average value of the product $f(x)f(y)$ for all pairs $(x, y) \in X^2(d)$. In terms of objective mean and variance, the formula becomes:

$$\rho(d) = \frac{1}{\sigma^2(f)|X^2(d)|} \sum_{(x,y)\in X^2(d)} (f(x) - \overline{f})(f(y) - \overline{f}). \tag{4.10}$$

The autocorrelation $\rho(d)$ can be estimated by computing a sufficiently large sample of solutions with distance d.

The Random Walk Correlation Function

Alternatively, Weinberger suggested to perform random walks to investigate the correlation structure of a landscape. The *random walk correlation function* [306, 278, 279]

$$r(s) = \frac{\langle f(x_t)f(x_{t+s})\rangle - \langle f\rangle^2}{\langle f^2\rangle - \langle f\rangle^2} \tag{4.11}$$

of a time series $\{f(x_t)\}$ defines the correlation of two points s steps away along a random walk through the fitness landscape.

If the landscape is *statistically isotropic*, i.e., the time series $\{f(x_t)\}$ forms a stationary random process, then a single random walk is sufficient to obtain $r(s)$:

$$r(s) \approx \frac{1}{\sigma_f^2\,(m-s)} \sum_{t=1}^{m-s} (f(x_t) - \overline{f})(f(x_{t+s}) - \overline{f}) \tag{4.12}$$

where m denotes the length of the random walk.

If a time series is *isotropic*, *Gaussian* and *Markovian* [306], then the corresponding landscape is called AR(1) landscape, and the random walk correlation function is of the form $r(s) = r(1)^s = e^{-s/\ell}$ with ℓ being the correlation length of the landscape. For example, AR(1) landscapes are found in the *NK*-model and the TSP [306].

The Correlation Length

Hence, the correlation length ℓ [279] of the landscape is defined as

$$\ell = -\frac{1}{\ln(|r(1)|)} = -\frac{1}{\ln(|\rho(1)|)} \tag{4.13}$$

for $r(1), \rho(1) \neq 0$. The correlation length directly reflects the ruggedness of a landscape: the lower the value for ℓ, the more rugged the landscape.

A ruggedness measure similar to the correlation length ℓ has been proposed in [9]. It is defined as:

$$\lambda = \frac{1}{1-\rho(1)} \approx \ell \tag{4.14}$$

and is called the *autocorrelation coefficient* λ [9].

It is useful to normalize the correlation length by the diameter of the landscape. Let

$$\xi = \frac{\ell}{\operatorname{diam} G_{\mathcal{L}}}. \tag{4.15}$$

With ξ close to 1, the landscape is highly correlated, and with ξ close to 0, there is no correlation.

Advanced Time Series Analysis

The analysis of a time series usually involves identifying an appropriate model that adequately represents the data generating process. Once the parameters of the model have been estimated, statistical tests can be applied to see how well the model approximates the data and what the explanatory and predictive value of the model is. Thus, a model of the form

$$y_t = g(y_t, y_{t-1}, \ldots, y_0) \tag{4.16}$$

is derived from the observed data, which can be used to predict the future values of a time series.

In [146], to measure and express the correlation structure of a fitness landscape, the Box-Jenkins approach [38] is proposed for finding a model for the time series $\{y_t\} = \{f(x_t)\}$ generated by a random walk trough a fitness landscape. The Box-Jenkins approach is a useful statistical method of model building. The purpose of the approach is to find an

ARMA model that combines an autoregressive (AR) model and a moving average (MA) model to represent a data generating process. An AR model of order p (AR(p)) has the form

$$y_t = \alpha_1 y_{t-1} + \ldots + \alpha_p y_{t-p} + \varepsilon_t, \tag{4.17}$$

with $\alpha_i \in I\!R$ and the stochastic variable ε_t representing white noise (mean zero and finite variance).

An MA model of order q (MA(q)) has the form

$$y_t = \varepsilon_t \beta_1 \varepsilon_{t-1} + \ldots + \beta_q \varepsilon_{t-q}, \tag{4.18}$$

where $\beta_i \in I\!R$ and the stochastic variable ε_t represents white noise with $Cov(\varepsilon_s, \varepsilon_t) = 0$. The combination of both yields the ARMA(p, q) model with

$$y_t = \alpha_1 y_{t-1} + \ldots + \alpha_p y_{t-p} + \varepsilon_t + \beta_1 \varepsilon_{t-1} + \ldots + \beta_q \varepsilon_{t-q}. \tag{4.19}$$

In other words, each value y_t is a weighted sum of p past values of the time series and q members of a white noise series.

There are three stages in the approach. First, an appropriate ARMA model is identified by finding the values for p and q utilizing a correlogram, in which $r(s)$ is plotted against s, and a partial correlogram, in which the partial correlation of the time series against s is plotted. Second, the parameters of the model are estimated, for example using the *coefficient of determination* as a measure of "goodness of fit". In the third and last step, the estimated model is tested for adequateness, for example using a higher-order model, e.g., for an AR(p) model a AR($p+1$) model is also estimated and the extra parameter (α_{p+1}) is shown to be not significant. A detailed description of the approach can be found in [147].

Selfsimilarlity

Another interesting property of fitness landscapes is the relation between correlation and self-similarity. A landscape is said to be *fractal* if the variance of the fitness difference between two points in the landscape scales as a power law with their distance from each other. More formally, a landscape is fractal [275] if

$$\langle |f(x) - f(y)|^2 \rangle \propto d(x,y)^{2h} \tag{4.20}$$

with $h \in (0,1)$ for all pairs of points (x,y) in the search space. This definition can be reformulated using the autocorrelation function [309] as

$$1 - \rho(d) \propto d^{2h}. \tag{4.21}$$

The definition of fractal landscapes is derived from the *fractional Brownian motion* (fBm). A fBm is a random function $f : I\!R^n \rightarrow I\!R$ with the distribution of $f(X')$ conditional on $f(X)$, X, and X' normally distributed with mean 0 and variance proportional to $||X' - X||^{2h}$ with $h \in (0,1)$. Such functions have the statistical scaling property that for any r, $f(rX)$ is statistically indistinguishable from $r^h f(X)$.

Examples of fractal landscapes with $h = \frac{1}{2}$ are *NK*-landscapes, TSP with edge–exchange, and the GBP [309].

4.5 Performance Prediction and Analysis for Evolutionary Algorithms

In the context of evolutionary computation, attempts have been made to find problem characteristics that make a problem hard for a EA. Furthermore, methods for analyzing evolutionary variation operators have been proposed.

4.5.1 Problem Difficulty

Several attempts have been made to find a measure for problem difficulty in the context of evolutionary algorithms (EAs). Certainly, there are many factors besides landscape ruggedness [168] that can influence the performance of an (evolutionary) search algorithm:

- *deception* [119]: Presence of sub-optimum solutions leading heuristic search algorithms away from the global optimum, which is isolated from the other near optimum solutions
- *multimodality* [148]: a high number of local optima
- *epistasis* [68]: the nonlinearity of the problem as a result of the interaction of the components of the solution vectors
- *noise* [256]: the effects of noise in fitness functions with signal and noise components
- *neutrality* [20]: the existence of terrains with equal fitness – neutral networks – in the fitness landscape

For example, Davidor suggested that epistasis variance is suitable for measuring problem difficulty for genetic algorithms (GAs) [68]. He argues that high epistasis – in other words: a high degree of chromosome interaction – characterizes problems that are hard for a GA. This implication can also be transferred to non–evolutionary search algorithms such as local search. Moreover, epistasis and landscape ruggedness have been shown to be related in NK-landscapes [168]: the higher the epistasis in a problem (the higher the K in the NK model), the more rugged the landscape of the instance. However, for problems for which a binary representation is not practical, the definition of epistasis is more complicated. For example, a definition of gene interaction is not obvious for permutation encodings.

There is a growing awareness of the importance of neutral mutations as a significant factor in evolutionary dynamics among population geneticists [172] and molecular biologists [84, 268]. Recently, inspired by Kimura's "Neutral Theory of Molecular Evolution" [172], the influence of the presence of neutral networks in fitness landscapes has been studied in the context of evolutionary optimization. Barnett [20] has shown that the structure of the neutral networks of a landscape can have higher influence on population dynamics than the correlation structure of a landscape.

4.5.2 Variation Operator Analysis

The random walk correlation analysis described above is well suited in studying the behavior of mutation-based EAs, since mutation operators can be thought of as producing a time series of fitness values for a sequence of solutions to an optimization problem. Generally, the higher the correlation, the better suited a mutation operator for a given landscape. Several attempts have been made to generalize this form of analysis to recombination operators in EAs.

In [197, 204], the parent child fitness correlation

$$\rho_{op} = \frac{Cov(F_p, F_c)}{\sigma(F_p)\,\sigma(F_c)} \tag{4.22}$$

has been used to compare crossover operators for GAs. Here, F_c denotes the average fitness of the children produced by crossover and F_p the average fitness of the parents.

In [146, 147], a time series is produced by repeated application of a crossover operator with a randomly generated mate, randomly chosen crossover point, and randomly selected child. The random walk correlation of the time series is computed, and the time series is further analyzed using the Box-Jenkins approach described above.

Stadler and Wagner [282] have extended the idea of viewing a landscape as a graph: a *P-structure* is a pair $(V, \mathcal{R})$ and consists of a vertex set and a mapping (recombination operator) $\mathcal{R} : V \times V \rightarrow \mathcal{P}(V)$. An algebraic theory for recombination spaces based on P-structures is proposed in [282].

4.6 Landscape Analysis and Memetic Algorithms

Although the above measures have been applied to genetic algorithms or simulated annealing, little effort has been made to use these or other techniques for hybrid evolutionary algorithms. For example, fitness distance correlation has been used to study genetic algorithms in [160], and it has been shown that there are counterexamples for which the FDC coefficient fails in predicting GA performance [5]. Correlation has been used in [197, 146, 147] to analyze mutation and crossover in genetic algorithms. Sorkin [275] has shown how fractal landscapes are related to the effectiveness of simulated annealing.

In memetic algorithms, the techniques for analyzing recombination and mutation operators only make limited sense due to the different search strategies inherent in MAs.

4.6.1 The Role of Evolutionary Operators in Memetic Algorithms

In understanding how memetic algorithms work, the concept of fitness landscapes appears very useful. Since all individuals in the population of a MA, as described in the previous chapter, represent local optima (which is achieved by applying local search as needed) we can think of the memetic search as follows: The MA produces a set of locally optimum solutions by applying the evolutionary variation operators recombination and/or mutation to the current population of local optimum solutions. Selection then reduces the set of individuals to the original size of the population. Thus, the role of the variation operators is to discover other local optima with higher fitness. Figure 4.1 illustrates the 'jumps' performed by the operators in a landscape of a minimization problem. Thus, a jump has to be far enough to escape from the basin of attraction of the current local optimum. If the jump is too far, the search may degenerate to a multi–start local search that does not exploit the structure in the distribution of the local optima (assuming that a structure is present). The local search is responsible for descending from the destination point of the jump into the local optimum, as illustrated in the figure.

From this picture it becomes obvious that mutation and recombination operators used in simple evolutionary algorithms are not always suited in MAs. A mutation operator with a

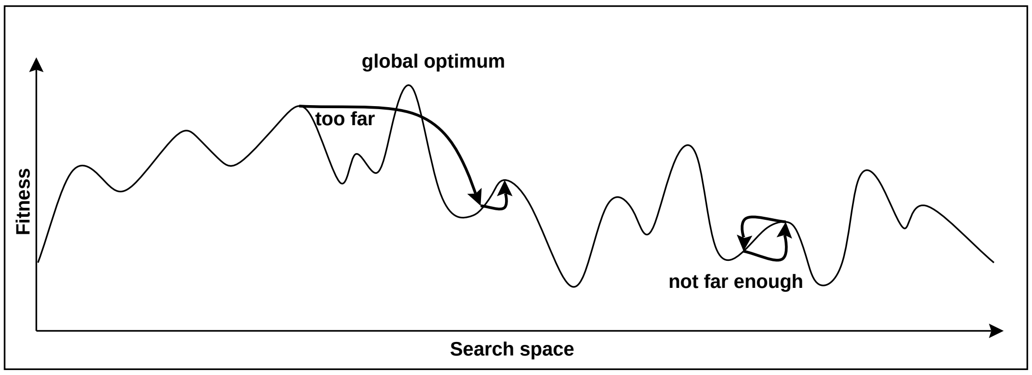


Figure 4.1: Variation Operators and the Fitness Landscape

low mutation rate as usually used in GAs would have no effect since the mutation would be reversed by the local search. On the other hand, the recombination operator does not need to produce offspring with high fitness (the fitness of parent and offspring are not required to have a high correlation). Instead, the operator just has to produce offspring in the attractor region of a local optimum with high fitness. Hence, the methods for analyzing mutation and recombination operators in standard evolutionary algorithms are not appropriate for MAs.

Mutation

Mutation is an unary operator and has to be performed so that the subsequently applied local search does not revisit the parent solution by falling back into the same local optimum. The optimal mutation rate and thus resulting jump distance depends at least on one property of the search space: the size of the attractor region of the current local optimum. Assuming that on average one local optimum is contained in a ball of radius R, an effective mutation operator has to produce solutions with a distance greater than R in order to minimize the probability to fall back into the same local optimum. In some cases, however, lower mutation rates are reasonable due to properties of the neighborhood used in the local search, as will be shown in chapter 7. Generally, the mutation operator should be designed to produce solutions that are not in the neighborhood on which the local search is based.

Recombination

Recombination is an operator that requires two or more solutions as arguments. Hence, jumps resulting from the application of recombination operators can be performed with a predefined direction.

Consider the recombination operators k-point crossover and uniform crossover described in the previous chapter. These crossover techniques applied to binary vectors have the following properties:

(i) The alleles that are identical in both parents are preserved in the offspring.

(ii) The offspring contain only alleles from either one of the parents.

Following Radcliffe's and Surry's terminology [255, 254], a recombination operator obeying (i) is called *respectful*, while an operator obeying (ii) is called *assorting*. With respect to the fitness landscape, the first property implies that the Hamming distance d_H between the parents x and y and the offspring z are lower or equal to the distance between the parents. The second property implies that the offspring lie on a shortest path between the two parents and hence the equality $d_H(x, z) + d_H(z, y) = d_H(x, y)$ holds.

For permutation search spaces, the first property can easily be fulfilled while the second can not. However, if (i) is obeyed, the following holds for all parents x and y and offspring z: $d(z, x) \leq d(x, y)$ and $d(z, y) \leq d(x, y)$, and furthermore $d(x, z) + d(z, y) \leq 2 \cdot d(x, y)$. Recombination operators which fulfill (i) or even (ii) produce offspring that are contained in

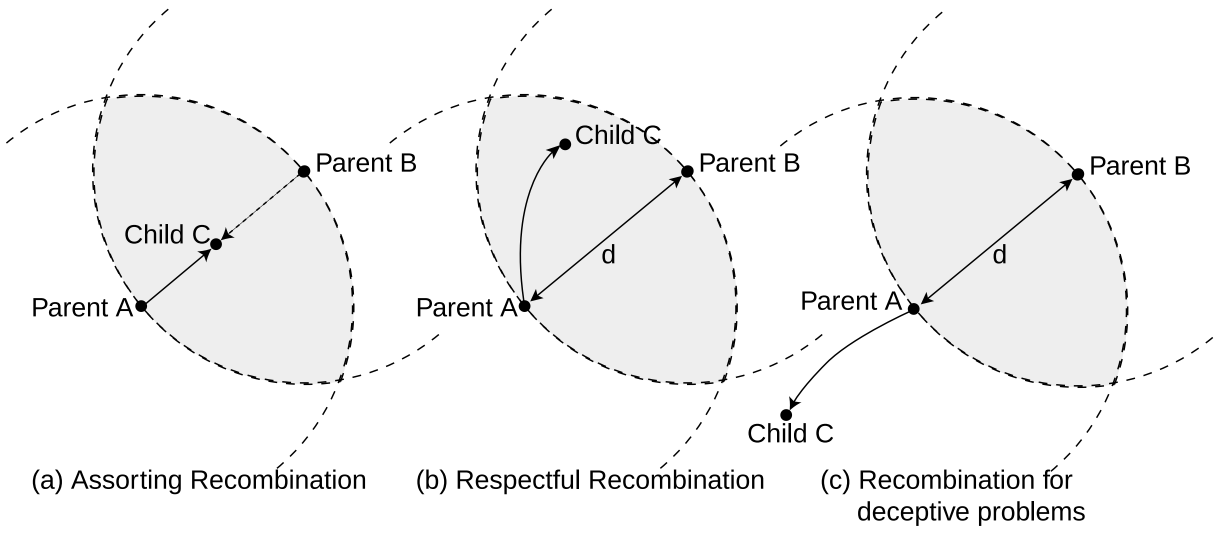

Parent B
Child C
Parent A
(a) Assorting Recombination
Child C
Parent B
d
Parent A
(b) Respectful Recombination
Parent B
d
Parent A
Child C
(c) Recombination for deceptive problems

Figure 4.2: Geometric Visualization of Recombination Operator Behaviors

a region of the search space spanned by the two parents. In Figure 4.2, two types of respectful recombination operators are shown. The first one (a) is assorting, while the second (b) is not. From the figure it can be seen that recombination produces directed jumps in the search space, i.e., from one parent solution towards the other parent solution. Thus, if a jump into the region between two local optima yields local optima with higher fitness on average than jumps of the same distance in an arbitrary direction, recombination is preferable to mutation. If a jump into a region between two local optima leads to local optima with worse fitness than arbitrary jumps of the same distance, the problem is called deceptive. In such a case, a recombination operator can be defined that exploits this characteristic of the fitness landscape. Figure 4.2(c) shows a recombination operator for deceptive problems. Assuming that parent B has a higher fitness than parent A, the operator produces solutions which are distant from the subspace defined by the common features of the two parents and which have a higher distance to the parent with better fitness. Obviously, such an operator is *not* respectful.

Population Dynamics

The population in a memetic algorithm can be regarded as a set of points moving in the fitness landscape. Initially, the points are distributed over a large fraction of the search space, sometimes even the whole fitness landscape depending on the method used to generate the starting population of local optima. Then, during evolution, the points tend to move closer to each other, concentrating in a small fraction of the search space. In a genetic algorithm scenario where assorting recombination is exclusively used, the average distance of the population converges exponentially fast towards zero, since in each generational step the distance of the individuals is halved. This behavior can be also observed in many MAs: the average distance of the population decreases rapidly and the points in the search space move towards each other. Thus, a series of jumps towards other solutions is performed during the run, where the jump distance decreases dynamically when using respectful recombination. In an evolutionary algorithm, convergence is a necessary effect, since from an information theory point of view, the loss of diversity can be viewed as "information gain".

Using the restart mechanism as described in the previous chapter, the process of contraction is repeated and the exploitation of other regions is enforced as shown in the figure. In a MA employing only mutation to achieve variation, this behavior can also be observed for certain landscapes. The average distance of the population can be expected to decrease at least to the average distance of the local optima in the landscape. However, there are landscapes for which a MA does not converge at all.

4.6.2 Landscape Analysis and the Choice of Operators

In order to design a memetic algorithm for a combinatorial problem, i.e., to decide which local search and which variation operators are best suited, it is required that the fitness landscape is analyzed. In a first step, we are interested in finding the best suited local search.

Finding Local Optima

The ideal properties of a local search for a MA are the following. The local search should

- be fast,
- produce near optimum solutions,
- introduce only few local optimum solutions, and
- be efficient in searching a limited region of the search space.

However, a tradeoff between running time of the local search and thus exploitation time versus exploration time of the evolutionary framework has to be made. The longer the running time of the local search, the less local optimum solutions can be visited.

To compare neighborhoods for a problem, the landscapes they define can be analyzed by calculating or estimating the *correlation length* of the landscape. Generally, the more correlated the landscape, the more effective the local search. The number of iterations to reach a local optimum is usually higher in correlated landscapes due to larger basins of attractions and less local optima. Thus, the exploitation of a region of the search space is often higher and the produced optima are better than for a local search with few iterations. An advantage of the landscape (auto)correlation analysis is that the correlation can be computed mathematically for some problems. It has been shown, for example, that in the TSP, local search neighborhoods based on edge exchange lead to much higher correlated landscapes than neighborhoods based on city exchanges [281]. Not surprisingly, the former local search algorithms perform much better than the latter.

The quality of the solutions can be compared in experiments by running the local searches. Additionally, it is useful to calculate the average distance of the starting solutions and the resulting local optima. The distance is low if local search stays in a limited region of the search space. If the distance is high, then the local search does not exploit regions in the search space and traverses large portions of the search space directionless.

The Choice of Variation Operators

Once the best local search has been identified, the search space is analyzed to determine the variation operators for the MA. The effectiveness of the evolutionary operators highly depends on the distribution of the local optima in the search space. To identify a structure in the distribution of the local optima, a fitness distance analysis (FDA) is well-suited. In comparison to the estimation of the FDC coefficient described above, the correlation of fitness and distance to the optimum is investigated for the local optima. Information about the distribution of the local optima is drawn from the fitness distance plot, in which the fitness of the local optima is plotted against their minimum distance to an optimum. Instead of plotting fitness, the difference of objective value to the optimum solution value can be used ($\Delta f = |f_{opt} - f(x)|$), so that the plots of similar landscapes of a maximization and a minimization problem have the same appearance. The FDC coefficient can also be computed but plays just a secondary role. The plot provides more insight, since it contains more information. Several researchers have used such FD plots for analyzing local optima, including Kauffman [168] for *NK*-landscapes, Boese [35] for the TSP, and Reeves [259] for a flow–shop scheduling problem, and Merz and Freisleben [217] for the graph bipartitioning problem.

A major weakness of the FDC analysis is that the optimum solution has to be known in advance. However, in most cases a near optimum solution found in experiments can be used to replace the optimum solution. For correlated and uncorrelated landscapes, the FD plots look similar to plots with the distance to the optimum solution.

Sometimes, it appears to be useful to generate another plot during analysis: the average distance between the local optima against the distance to the optimum reveals additional information how the local optima are located relatively to each other in the search space. Additionally, when performing FDA, it is useful to calculate other properties such as the number of distinct local optima found, and the average distance between the starting solutions and the local optima as noted before.

The right choice of a variation operator for a MA depends on the distribution of the local optima. Consider three different landscapes for which the fitness distance plots look as displayed in Figure 4.3. The plot in the upper left shows the distribution of Lin-Kernighan local optima of a TSP instance. The plot reveals that the local optima are found only in a small fraction of the search space (the maximum distance to the optimum is approximately a third of the diameter of the landscape), and fitness and distance are correlated. Thus, a respectful recombination scheme can be used to exploit this structure. An example of a totally uncorrelated landscape is shown in the upper right. The local optima of the *NK* landscape have a distance up to the maximum distance to the optimum as well as to each other (which can be verified by generating a mean distance - distance to optimum plot). Here, a recombination scheme is of little help. Instead, a mutation operator with relatively high, constant jump distance is more appropriate. The plot in the lower left shows a very high correlation of the Kernighan-Lin local optima of a regular GBP instance. This landscape has

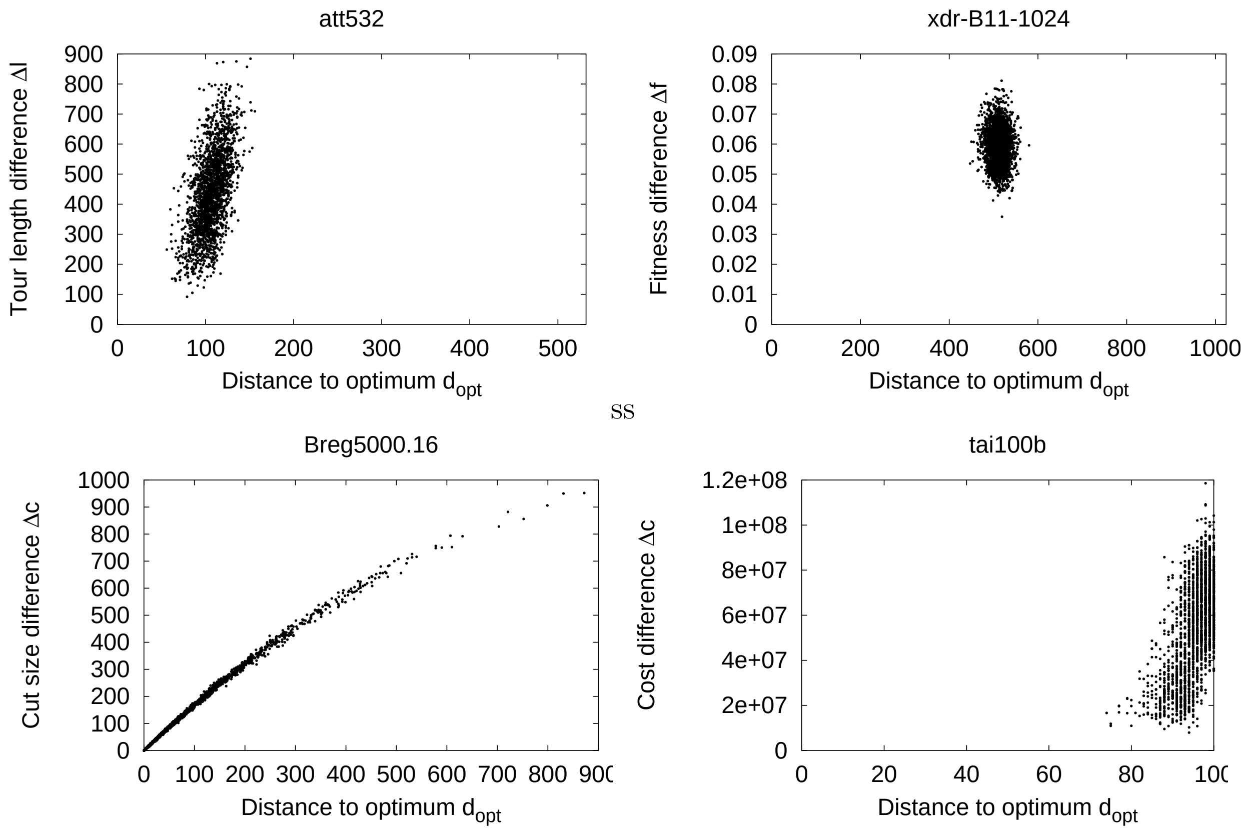


Figure 4.3: Correlation of local optima of a TSP, *NK*-model, GBP, and QAP instance

an ideal distribution of local optima. The optimum solution can be found by "jumping" from one optimum to a better one with successively decreasing the jump distance. Furthermore, the optimum lies central among the local optima such that a recombination operator with high tendency to produce offspring on a shortest path between other local optima with near optimal fitness is best suited. Finally, the plot in the lower right shows the correlation of local minima of a QAP instance. All local optima have a large distance to the local optimum but a slight tendency of decreasing cost difference with decreasing distance to the best-known solution can be observed.

The Choice of Greedy Components

The use of randomized greedy heuristics for generating starting solutions depends highly on the problem structure. Greedy heuristics are especially effective if the problem has low epistasis, since the greedy choices of a decision variable affect only few other choices. In the extreme case, where there is no linkage between the genes, a greedy heuristic will always find the optimum solution. In the other extreme, if all genes (or decision variables) depend on all other genes, the greedy heuristic does not perform better than random search.

To analyze a greedy heuristic, a fitness distance plot can be utilized for visualizing the distribution of the greedy constructed solutions. The randomized greedy heuristic should be capable of producing a variety of different solutions so that the search space is adequately sampled.

In a MA for a problem with low epistasis, greedy choices can be easily incorporated in the recombination procedure. A respectful greedy recombination operator first copies all alleles that are common in both parents and then uses a greedy construction procedure to produce a feasible solution from the partial solution.

4.7 Summary

In this chapter, the performance evaluation and the performance prediction of heuristic optimization approaches have been addressed. The most important issue in the research area of heuristic optimization techniques including evolutionary computation is gaining insight into the operation of these techniques. When we know *why* an algorithm (or an instance of an algorithm) performs well on a type of problem, we are able to predict its performance by identifying the type of problem to solve. Thus, we become able to choose the best performing among the alternative instances of an algorithm. To gain insight, the notion of fitness landscapes has been introduced that provides an intuitive picture of how a heuristic search algorithm works. The search space can be regarded as a spatial structure in which a heuristic performs moves from one point to another in order to find solutions of high quality. Statistical properties of these landscapes can be found by a landscape analysis that provides links between types of problems and search strategies that are best suited. Several landscape properties and methods of their analysis have been discussed in this chapter. Those especially important for the design of highly effective memetic algorithms have been addressed in more detail. In MAs, regions of the search space are exploited by a local search procedure that efficiently visits points of high fitness using a predefined neighborhood structure. Autocorrelation or random walk correlation analysis can help to identify the most appropriate neighborhood: the more correlated the neighboring points in the search space, the lower the landscape ruggedness and hence the more effective a local search.

Memetic algorithms can be designed to follow several search strategies depending on the variation operators to explore the search space. Variation operators can be regarded as performing "jumps" in the search space to escape from the attractor region of a local optimum with the aim of reaching the basin of attraction of a local optimum with higher fitness. Variation operators can be divided into mutation and recombination operators, the latter can be further divided into unrespectful, respectful and assorting recombination operators. The effectiveness of the operators highly depends on another property of the landscape: the distribution of the local optima (with respect to the chosen local search) within the search space. If there is a certain structure in the arrangement of the local optima, e.g. the fitness and distance to the optimum are correlated, a recombination operator can be used to exploit this feature. If, on the other hand, the locally optimum solutions are uniformly distributed in search space, a mutation operator with a sufficiently large jump distance to escape the attractor regions of local optima is a better choice. The fitness distance (correlation) analysis of local optima provides a way to "measure" the arrangement structure of the locally optimum solutions. A fitness distance plot is well suited in identifying characteristics: if the problem is deceptive and therefore the plot shows a positive correlation (the higher the fitness, the higher the distance to the optimum), an unrespectful recombination operator can be used to exploit this property. For high, negative correlations, assorting or simple respectful recombination operators should be used.

Moreover, it has been argued that fitness distance analysis can be utilized for determining the usefulness of greedy choices (borrowed from greedy heuristics) in the initialization and recombination steps.

Chapter 5

NK-Landscapes

5.1 Introduction

The *NK*-model of fitness landscapes has been introduced by Kauffman [168] to study gene interaction in biological evolution. In the *NK*-model, the fitness is the average value of the fitness contributions of the loci in the genome. For each locus, the fitness contribution is a function of the gene value (allele) at the locus and the values of K other interacting genes. Although this model is a very simplified model, it allows to produce families of fitness landscapes with interesting properties.

Besides its biological implications, the model is interesting for researchers in the field of evolutionary computation, since the *NK*-landscape model provides combinatorial optimization problems with tunable difficulty. Furthermore, traditional genetic algorithms can be applied easily to these problems, since a genome in the *NK*-model is represented by a binary string of fixed length.

In this chapter, new greedy and *k-opt* local search heuristics for *NK*-landscapes are proposed that can be easily embedded into memetic algorithms. The properties of *NK*-landscapes are discussed and a fitness distance correlation analysis is performed for the newly introduced heuristic algorithms. Based on the results of the analysis, the performance of memetic and even simple evolutionary algorithms can be predicted: For low epistasis – low values of K in the model – recombination based algorithms are able to exploit the structure of the search space effectively. With increasing epistasis, the landscapes become quickly unstructured, limiting the usefulness of recombination. For high epistasis, mutation based algorithms become favorable over recombination based evolutionary algorithms.

In computer experiments it is shown that simple genetic algorithms based on recombination or mutation do not scale well with increasing problem size, although *NK*-landscapes offer perfect conditions for the application of GAs. In a comparison study it is shown that simple MAs are able to outperform GAs based on crossover and EAs based on bit-flip mutation.

Finally, the effectiveness of more sophisticated MAs based on the proposed greedy and *k-opt* local search heuristics is demonstrated. These algorithms offer (near) optimum solutions in short time even for high dimensional landscapes.

The results presented in this chapter have been parially published in [212].

5.2 Heuristics for the *NK*-Model

Since *NK*-Landscapes have only been studied in the context of simulated biological evolution, little attention has been payed to the development of simple non-evolutionary heuristics. However, besides hill climbing/local search techniques, constructive heuristics such as the greedy algorithm can be applied to problems of the *NK*-model.

In the following, a solution vector x is assumed to be a binary vector of length N ($x = (x_1, \ldots, x_N)$) with the fitness

$$f(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x_i, x_{i_1}, \ldots, x_{i_K}), \tag{5.1}$$

where the fitness contribution f_i of locus i depends on the value of gene x_i and the values of K other genes $x_{i_1}, \ldots, x_{i_K}$. The function $f_i : \{0,1\}^{K+1} \to I\!R$ assigns a uniformly distributed random number between 0 and 1 to each of its 2^{K+1} inputs.

5.2.1 Greedy Algorithms

A point in a *NK*-landscape can be constructed in N steps by assigning in each step a gene value to a gene at a given locus. If the choice of a gene value follows a greedy rule, such an approach can be classified as a greedy heuristic for *NK*-landscapes.

The greedy heuristic proposed in this thesis works as follows. A solution is built in N steps by choosing a gene which is still not assigned a value, and a gene value to assign to the gene. The choice is made by maximizing a gain function $g(i,v) : \{1, \ldots, N\} \times \{0,1\} \to I\!R$ with $g(i,v)$ denoting the gain attained by setting the value of the i-th gene to v. The gain function $g(i,v)$ is defined as the difference between the fitness of a partial solution y with gene i set to v and the fitness of a partial solution x with gene i unspecified: $g(i,v) = f^p(y) - f^p(x)$ with

$$y_j = \begin{cases} v & \text{, if} \quad i = j \\ x_j & \text{, otherwise.} \end{cases}$$

The fitness f^p of a partial solution is defined as the average fitness of all solutions matching the template defined by the partial solution: Assume the partial solution x is $x = (1, 0, *, 0, *, 1)$ with $*$ denoting the *don't care* symbol (the gene has no value). Then, the fitness f^p of x is the average fitness of the four solutions $(1,0,\underline{0},0,\underline{0},1)$, $(1,0,\underline{0},0,\underline{1},1)$, $(1,0,\underline{1},0,\underline{0},1)$, and $(1,0,\underline{1},0,\underline{1},1)$.

Assuming the fitness contribution of site i denoted $f_i(x_i, x_{i_1}, \ldots, x_{i_K})$, depends on the site i itself and K neighbors $i_1, \ldots, i_K$, then the neighborhood $N_i = \{i, i_1, \ldots, i_K\}$ defines the set of genes/loci which contribute to the fitness at site i. The set of loci/genes which depend on the value of gene k is thus defined as $D_k = \{i \mid k \in N_i\}$.

Hence, the gain function becomes

$$g(i,v) = f^p(y) - f^p(x) = \sum_{i \in D_k} f_i^p(x_i, \ldots, v, \ldots) - f_i^p(x_i, \ldots, x_k, \ldots). \tag{5.2}$$

Initially, the partial fitness contribution of locus i is the average over all 2^{K+1} possible values of f_i. Hence, the greedy heuristic based on partial fitness calculations requires more than $n \cdot 2^{K+1}$ additions and is therefore only practically useful for landscapes with small values of K. On the other hand, with increasing K, the solutions produced by the greedy heuristic approach the average fitness of the points in the landscape since for high epistasis the values of f_i^p differ significantly from the values of f_i in the final solution.

5.2.2 Local Search

The application of local search techniques to *NK*-landscapes is straightforward: Neighboring solutions can be reached by flipping one or more bits simultaneously in the genome. However, instead of calculating the fitness for each neighboring solution anew it is more efficient to calculate the gain achieved by moving to the new solution. In this context the gain is referred to as the fitness difference between the new and the old solution.

The gain associated with the flipping of a single gene k in the genome x leading to a solution y with

$$y_i = \begin{cases} 1 - x_i & \text{, if} \quad i = k \\ x_i & \text{, otherwise} \end{cases}$$

is the fitness difference of the new solution y and the old solution x:

$$g_k(x) = f(y) - f(x) = \sum_{i \in D_k} f_i(x_i, \ldots, 1 - x_k, \ldots) - f_i(x_i, \ldots, x_k, \ldots). \tag{5.3}$$

A local search for the *NK*-model can be implemented by maintaining a gain vector $g = (g_1 \ldots, g_N)$ instead of calculating all gains anew in each iteration. After flipping gene k, generally not all of the gains have to be updated. A gain g_i only changes if there is a $j \in D_i$ with $k \in N_j$ or in words the gain of flipping gene i changes if there is a fitness distribution function that depends on the value of gene k and i.

A *1-opt* Local Search

A simple local search based on a 1-opt neighborhood can be realized straightforwardly. The neighborhood is searched by flipping a single bit in the current solution. The gain vector can now be used to find an improving flip in reduced computation time. However, after flipping the gene value, some elements of the gain vector have to be updated accordingly. The pseudo code for such a local search is displayed in Figure 5.1. Alternative to selecting

```
procedure Local-Search-1-opt($x \in X$): $X$;
  begin
    calculate gains $g_i$ for all $i$ in $\{1, \ldots, n\}$;
    repeat
      find $k$ with $g_k = \max_i g_i$;
      if $g_k > 0$ then
        $x_k := 1 - x_k$;
        update gains $g_i$;
      endif
    until $g_k \leq 0$;
    return $x$;
  end;
```

Figure 5.1: Fast 1–opt Local Search for *NK* Landscapes

the best move in each iteration, a move can be selected randomly among those with positive associated gain.

A *k-opt* Local Search

The basic scheme described above can be extended to derive more powerful local search algorithms. For example, a *2-opt* local search can be realized by flipping two genes to reach a solution in the neighborhood of the current solution. More generally, a *k-opt* local search can be realized by flipping k genes simultaneously. Since the neighborhood size of a *k-opt* local search grows exponentially with k, mechanisms are required to perform a *k-opt* local search in reasonable time. This can be achieved be considering a small fraction of the *k-opt* neighborhood similarly to the heuristics by Lin and Kernighan for the TSP [192] and the GBP [171]. The *k-opt* local search for *NK*-landscapes proposed in this thesis is based on the ideas of Lin and Kernighan: in each iteration, a variable number of genes is flipped, depending on a gain criterion. To find the most profitable *k-opt* move, a sequence of up to n solutions is generated by stepwise flipping genes with the highest associated gain. Every gene is flipped no more than once to guarantee that all solutions in the sequence are different. The solution in the sequence with the highest gain is accepted as the new current solution. This solution may differ in 1 up to n genes depending on the position in the sequence. The pseudo code for the approach is provided in Figure 5.2. To reduce the running time of the algorithm,

```
procedure Local-Search-k-opt($x \in X$): $X$;
  begin
    calculate gains $g_i$ for all $i$ in $\{1, \dots, N\}$;
    repeat
      $x_{prev} := x$, $G_{max} := 0$, $G := 0$, steps $= 0$, $C := \{1, \dots, N\}$;
      repeat
        find $j$ with $g_j = \max_{i \in C} g_i$;
        $G := G + g_j$;
        $x_j := 1 - x_j$;
        if $G > G_{max}$ then
          $G_{max} := G$;
          $x_{best} := x$;
        endif
        update gains $g_i$ for all $i$;
        $C := C \backslash \{j\}$;
        $steps := steps + 1$;
      until $steps > maxsteps$ or $C = \emptyset$;
      if $G_{max} > 0$ then
        $x := x_{best}$;
      else
        $x := x_{prev}$;
      endif
    until $G_{max} \leq 0$;
    return $x$;
  end;
```

Figure 5.2: *k-opt* Local Search for *NK* Landscapes

the value for the maximum k can be bound to a value smaller than N. Furthermore, the inner repeat loop may be terminated if there was no new x_{best} for more than m solutions.

The *k-opt* local search described here can be applied to any binary-coded problem, in fact the *k-opt* local search proposed for the BQP in this thesis is very similar to the one described above. The gains g_i, however, are calculated differently.

5.3 The Fitness Landscape of the *NK*-Model

The *NK*-model of Kauffman [168, 169] defines a family of fitness landscapes which can be tuned by two parameters: N and K. While N determines the dimension of the search space, K specifies the degree of epistatic interactions of the genes constituting a genome. Each point in the fitness landscape is represented by a bit string of length N and can be viewed as a vertex in the N-dimensional hypercube.

With this model, the "ruggedness" of a fitness landscape can be tuned by changing the value of K and thus the number of interacting genes per locus. Low values of K indicate low epistasis and high values of K represent high epistasis.

To get a feeling of how the various landscapes look like, it is helpful to look at the two extreme cases of the *NK*-model which are the landscapes with $K = 0$ and the landscapes with $K = N - 1$.

Properties of $K = 0$ Landscapes

The $K = 0$ landscapes have the following properties [168]:

- There is only one local/global optimum
- The landscape is smooth; neighboring points (*1-opt* neighbors) in the search space are highly correlated. The fitness of 1-opt neighbors can differ by no more than $\frac{1}{N}$.
- The number of fitter neighbors decreases by one in each iteration of a 1-opt local search.
- The average number of iterations to reach the optimum is $\frac{N}{2}$ and thus in $O(N)$.

For the highest value of K, the properties of the fitness landscapes become quite different.

Properties of $K = N - 1$ Landscapes

If $K = N - 1$, the fitness contribution of a gene depends on the values of all other genes, which results in a highly uncorrelated, rugged fitness landscape. These landscapes have the following properties [168]:

- The expected number of local optima is $\frac{2^N}{N+1}$
- The expected fraction of fitter 1-opt neighbors dwindles by $\frac{1}{2}$ after each iteration of a 1-opt local search
- The number of improvement steps in a 1-opt local search to reach the local optimum is expected to be in $O(\log N)$
- The expected number of solutions to examine to reach a 1-opt local optimum is proportional to N

- The ratio of accepted to tried moves scales as $\log N/N$
- Starting from an arbitrary solution, only a small fraction of local optima can be reached by a 1-opt local search. An upper bound is given by the formula $N^{\log_2(N-1)/2}$
- Only from a small fraction of starting solutions $(2^{(\log_2 N)^2/2})$, the global optimum can be reached by 1-opt local search.

Furthermore, Kauffman [168] has shown that for increasing N, the fitness values of the local optima decrease towards $\frac{1}{2}$. He calls this phenomenon *a complexity catastrophe.*

Random vs. Adjacent Neighbor Model

Besides the values for the parameters N and K, the choice of the neighbor model is important for *NK*-landscapes, too. Kauffman [168] distinguishes two variants, the *random neighbor model* and the *adjacent neighbor model.* In the former, the genes which contribute to the fitness at locus i are chosen at random. In other words, the neighbors i_1 through i_K are randomly selected among the N. In the latter, the i_1 through i_k are the nearest loci to the gene at locus i, e.g. if $K = 2$ and $i = 5$, then $i_1 = 4$ and $i_2 = 6$, the neighborhood is $N_5 = \{5, 4, 6\}$.

The landscape properties described above are independent of the neighbor model. However, Weinberger [308] has shown that the computational complexity of both models differs. He studied the *NK* decision problem – *Is the global optimum of a given instance of an NK landscape greater than some specified value F?* – and was able to show that the *NK* decision problem with adjacent neighbors is solvable in $O(2^K N)$ steps and is thus in $\mathcal{P}$ and that the *NK* decision problem with random neighbors is $\mathcal{NP}$-complete for $K \geq 3$. To the best of the authors knowledge, the question whether the *NK* decision problem with random neighbors is $\mathcal{NP}$-complete for $K = 2$ is open.

5.3.1 Autocorrelation Analysis

The random walk correlation function for the *NK*-model has been calculated by Fontana *et al.* [101]. Their results can be summarized as follows: The random walk correlation function becomes

$$r(s) \approx \left(1 - \frac{K+1}{N}\right)^s \tag{5.4}$$

for the adjacent and random neighbor model. Hence, the correlation length is in both cases

$$\ell \approx \frac{N}{K+1}. \tag{5.5}$$

It is not surprising that the correlation length decreases with increasing K.

Weinberger [307] derived formulas for the autocorrelation function of *NK*-landscapes. He found that the autocorrelation function $\rho(d)$ depends on the neighbor model of the landscape. In the random neighbor model, the autocorrelation function becomes

$$\rho(d) = \left(1 - \frac{d}{N}\right)\left(1 - \frac{K}{N-1}\right)^d, \tag{5.6}$$

and for the adjacent neighbor model, ρ becomes

$$\rho(d) = 1 - \frac{d(K+1)}{N} + \frac{d}{N\binom{N-1}{d-1}} \sum_{l=1}^{K} (K+1-l) \binom{N-l-1}{d-2}, \tag{5.7}$$

with d denoting the hamming distance between bit vectors.

The formula show that the *NK*-model allows to produce landscapes with arbitrary ruggedness. The correlation length can be set to 1 by choosing $K = N - 1$ leading to a totally random landscape with uncorrelated neighboring points. Choosing the other extreme $K = 0$, the correlation length grows to its maximum value: N resulting in a smooth, single peaked landscape.

5.3.2 Fitness Distance Correlation Analysis

In his studies of *NK*-landscapes, Kauffman [168] investigated the correlation of fitness and distance to the optimum of local optimum solutions with respect to *1-opt* local search. In this work, the analysis is extended by investigating fitness distance correlation with respect to the greedy heuristic and *k-opt* local search. Experiments were conducted for three selected instances with N fixed to 1024, K in $\{2, 4, 11\}$ and a random neighbor model. Since the optimum solutions for these instances are not known, the best solutions found with the MAs described below in long runs (14400 s on a Pentium II 300 MHz PC) are used instead. These solutions are likely to be the global optima or at least close to the global optima with respect to fitness and distance.

In the first experiment, the distribution of greedy solutions in the search space is investigated. The results of the analysis are summarized in Table 5.1. In the first column, the

Table 5.1: Results of the Fitness Distance Analysis of Greedy Solutions to three *NK* Landscapes

Instance	N	K	$\min d_{opt}$	$\overline{d}_{opt}$	$\overline{d}_{gr}$	N_{gr}	ϱ
C2-1024	1024	2	130	220.62 (0.22)	195.03	2500	-0.62
D4-1024	1024	4	264	372.29 (0.36)	377.38	2500	-0.24
B11-1024	1024	11	458	515.74 (0.50)	469.35	2500	-0.01

name of the instance is displayed, and in the second and third column the parameters N and K are given. In columns four through eight, the minimum distance of the greedy solutions to the expected global optimum ($\min d_{opt}$), the average distance of greedy solutions to the global optimum ($\overline{d}_{opt}$), the average distance between the greedy solutions ($\overline{d}_{gr}$), the number of distinct greedy solutions (N_{gr}) out of 2500, and the fitness distance correlation coefficient (ϱ) are provided, respectively. Additionally, the normalized average distance, i.e. the average distance of the local optima to the global optimum divided by the maximum distance in the search space N is shown in column five in parentheses.

For small K, the greedy solutions are close to each other and close to the best known solution. There is a correlation between fitness and distance to the best known solution as the value ρ indicates. About three fourth of the gene values are equal in all greedy solutions for $K = 2$ and thus the solutions are contained in a small fraction of the search space. With increasing K, average distance between the greedy solutions quickly converges to the average distance ($N/2$) of the solutions in the search space. Surprisingly, already at $K = 11$ there is no correlation between greedy solutions and they have random distribution in the search space as expected for large values of K. In the second and third experiment, the correlation of

Table 5.2: Results of the Fitness Distance Analysis of *1-opt* Solutions to three *NK* Landscapes

Instance	N	K	$\min d_{opt}$	$\overline{d}_{opt}$	$\overline{d}_{loc}$	N_{1-opt}	ϱ
C2-1024	1024	2	215	337.21 (0.33)	381.75	2500	-0.59
D4-1024	1024	4	383	463.56 (0.45)	489.25	2500	-0.27
B11-1024	1024	11	447	511.47 (0.50)	511.90	2500	0.03

fitness and distance to the best known solution of *1-opt* and *k-opt* solutions was investigated. The results are shown in Tables 5.2 and 5.3. Again, in the first column, the name of the instance is displayed, and in the second and third column the parameters N and K are given. In columns four through eight, the minimum distance of the locally optimum solutions to the expected global optimum ($\min d_{opt}$), the average distance of the local optima to the global optimum ($\overline{d}_{opt}$), the average distance between the local optima ($\overline{d}_{loc}$), the number of distinct local optima (N_{1-opt}, N_{k-opt}) out of 2500, and the fitness distance correlation coefficient (ϱ) are provided, respectively. Additionally, the normalized average distance, i.e. the average distance of the local optima to the global optimum divided by the maximum distance in the search space N is shown in column five in parentheses. Similar as for the greedy heuristic,

Table 5.3: Results of the Fitness Distance Analysis of *k-opt* Solutions to three *NK* Landscapes

Instance	N	K	$\min d_{opt}$	$\overline{d}_{opt}$	$\overline{d}_{loc}$	N_{k-opt}	ϱ
C2-1024	1024	2	191	301.47 (0.29)	346.16	2500	-0.65
D4-1024	1024	4	347	440.57 (0.43)	470.36	2500	-0.33
B11-1024	1024	11	459	511.88 (0.50)	511.72	2500	0.02

the average distance between the local optima and the average distance to the best known solution increases quickly with increasing K. At $K = 11$ there is no correlation between fitness and distance, and the distribution is similar to a uniform distribution of random points in the search space. There is slightly higher correlation in case of *k-opt* in comparison to *1-opt* in case of the $K = 2, 4$ landscapes. However, greedy solutions have even a shorter minimum and average distance to the best known solution than *k-opt* solutions. The fitness distance plots for the three instances are shown in Figure 5.3. On the left, the scatter plots for the greedy solutions are provided, and on the right the scatter plots for *k-opt* solutions are displayed. The *2-opt* plots are not shown since they are very similar to the plots of *k-opt* solutions. For $K = 2$, the orientation of the points towards the origin is obvious. The cloud of points "moves" with increasing K quickly to the middle of the plane losing the orientation to the origin and thus to the optimum. These results correspond to the findings of Kauffman [168]. He further observed that for instances of the adjacent neighbor model the correlation of fitness and distances decreases not as rapidly as for the random neighbor model with increasing K.

From the perspective of performance prediction of MAs, the analysis provides some useful information. For small K (< 5) , recombination-based memetic algorithms are expected to

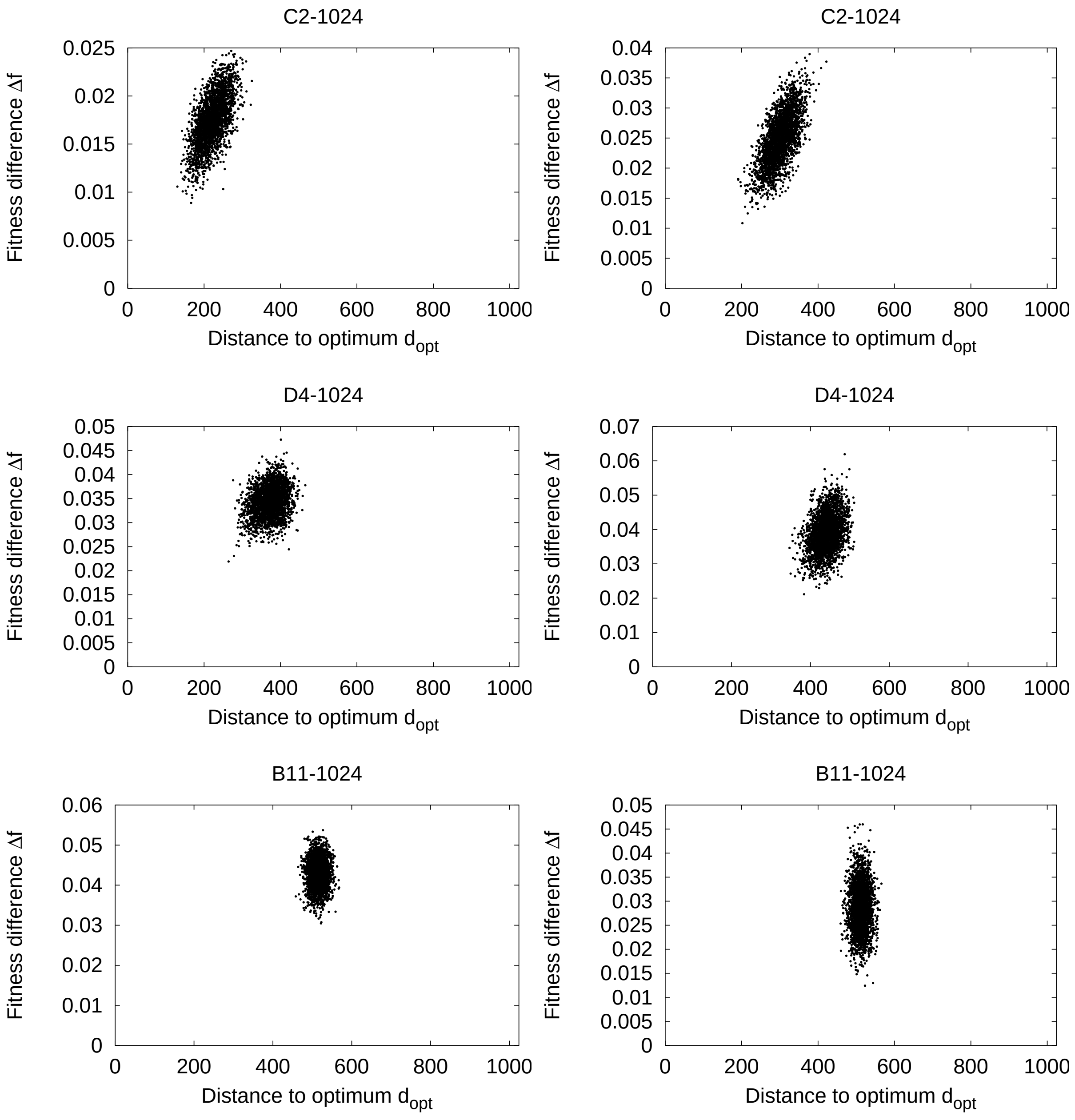


Figure 5.3: Fitness-Distance Plots of Greedy Solutions (left) and *k-opt* Solutions (right)

have a good performance since with recombination the fitness distance correlation of the local optima can be exploited: With increasing fitness, the local maxima are closer together, and their distance to the optimum becomes smaller. Furthermore, the locally optimum solutions are found in a small region of the search space in which the global optimum has a more or less central position. The greedy heuristic is very well suited for these instances with low epistasis and it is therefore promising to include the heuristic in the initialization phase of the population as well as in the recombination step. For larger K, the effectiveness of recombination decreases and eventually mutation based EAs are better suited.

5.4 A Memetic Algorithm for *NK* Landscapes

The application of MAs to *NK*-landscapes is straightforward. Since problems of the *NK*-model are binary-coded, all GA operators known for bit strings can be used in a MA.

5.4.1 Population Initialization and Local Search

The population can be initialized by randomly generating bit strings and by subsequently applying local search. For low values of K, the use of the randomized greedy heuristic described above can be used alternatively in combination with local search.

Suitable local search algorithms are *1-opt* local search and *k-opt* local search as described above.

5.4.2 Evolutionary Variation Operators

Due to the binary coding of the problem, all operators on binary strings can be applied in an evolutionary algorithm and therefore in a memetic algorithm, such as single point or two-point crossover, uniform crossover and bit flip mutation operators.

Recombination

A variant of uniform crossover (UX) that is used in the CHC algorithm of Eshelman [88] is an alternative to the crossover operators noted above. The operator creates (with high probability) offsprings that have a maximum Hamming distance to the parents which is half of the distance between the parents themselves. The operator is called denoted *HUX* in the following.

Alternatively, the greedy construction scheme can be used in recombination to produce offspring. A *greedy recombination operator* denoted *GX* is therefore devised that works by first inheriting all the gene values that are common to the two parents to retain respectful recombination. Then the remaining loci are set making greedy choices as in the greedy heuristic described above. This operator is especially effective for problems with low epistasis.

Mutation

Simple bit flip mutation is not useful in a memetic algorithm, since the flipping of a single bit will be reversed by a subsequently performed local search with high probability. Hence more than one bit must be flipped simultaneously in the parent solution. If p bits are flipped by the mutation operator, the Hamming distance of the resulting offspring and the original parent solution becomes p. The value of p should be chosen to minimize the probability that the subsequent local search rediscovers the unmutated solution.

5.4.3 Performance Evaluation

Since optimum solutions are generally not known for *NK*-landscapes of arbitrary values of K and N, it cannot be determined whether a given algorithm has found the optimum or not. Thus, the relative performance of heuristic algorithms has to be investigated.

Table 5.4: Performance of six algorithms on 9 *NK*-landscapes

Alg.	**gen**	**Fitness**	**gen**	**Fitness**	**gen**	**Fitness**
	$K = 2, N = 12$		$K = 4, N = 12$		$K = 11, N = 12$	
GA-UX	74214	0.731330, 0.00%	73759	0.709572, 0.00%	68805	0.806116, 0.00%
GA-1X	77228	0.731330, 0.00%	76442	0.709572, 0.00%	70460	0.806116, 0.00%
GA-M	78830	0.731330, 0.00%	76733	0.709572, 0.00%	61691	0.806116, 0.00%
MA-C	256649	0.731330, 0.00%	188791	0.709572, 0.00%	80925	0.806116, 0.00%
MA-M	207388	0.731330, 0.00%	161325	0.709572, 0.00%	68589	0.806116, 0.00%
MLS	2813352	0.731330, 0.00%	2020097	0.709572, 0.00%	851995	0.806116, 0.00%
Best		0.731330 0.00%		0.709572 0.00%		0.806116 0.00%
Time:						120 s
	$K = 2, N = 64$		$K = 4, N = 64$		$K = 11, N = 64$	
GA-UX	92813	0.726979, 1.54%	90163	0.776360, 1.75%	73216	0.755914, 5.51%
GA-1X	116482	0.730261, 1.09%	111878	0.772472, 2.25%	86808	0.757258, 5.34%
GA-M	142796	0.731240, 0.96%	134603	0.780548, 1.22%	95112	0.762200, 4.73%
MA-C	161881	0.738320, 0.00%	93204	0.790226, 0.00%	17364	0.791819, 1.02%
MA-M	138577	0.738320, 0.00%	71474	0.790226, 0.00%	20023	0.792688, 0.91%
MLS	383323	0.738320, 0.00%	227791	0.790152, 0.01%	120469	0.787223, 1.60%
Best		0.738320, 0.00%		0.790226, 0.00%		0.8000067, 0.00%
Time:						300 s
	$K = 2, N = 256$		$K = 4, N = 256$		$K = 11, N = 256$	
GA-UX	32199	0.735637, 0.97%	29332	0.763774, 3.69%	20756	0.742701, 4.56%
GA-1X	44481	0.728916, 1.88%	39116	0.757387, 4.50%	25245	0.726255, 6.67%
GA-M	67639	0.735617, 0.98%	57011	0.766737, 3.32%	32735	0.749083, 3.74%
MA-C	35230	0.741997, 0.12%	15257	0.788822, 0.54%	752	0.760639, 2.25%
MA-M	34523	0.741688, 0.16%	16082	0.788270, 0.61%	3390	0.772114, 0.78%
MLS	23316	0.731418, 1.54%	13711	0.762318, 3.88%	5490	0.747768, 3.91%
Best		0.742879 0.00%		0.793068 0.00%		0.778175 0.00%
Time:						300 s

Genetic/Evolutionary Algorithms vs. Memetic Algorithms

In a first set of experiments, traditional genetic algorithms and evolutionary algorithms based on mutation are compared with memetic algorithms based on *1-opt* local search. The algorithms use either mutation or recombination operators to study their effectiveness on various landscapes.

The following algorithms were used in the experiments: The first algorithm (GA-UX) is a simple GA with uniform crossover [292]. In the second algorithm (GA-1X) the uniform crossover operator is replaced by one-point crossover [142]. The third algorithm (EA-M) is an EA with mutation: here, a single bit-flip mutation operator is used. These three algorithms use stochastic uniform selection for recombination and worst replacement ($\mu + \lambda$ selection) as the survival selection strategy. The crossover application rate was set to 0.5, i.e. $0.5 \cdot P$ new offsprings were created per generation (P denotes the population size). In the GA-M algorithm, $0.2 \cdot P$ offsprings were created per generation with a mutation rate of $\frac{1}{N}$ per bit. Together with the restart technique described in chapter 3, the algorithms are similar to the CHC algorithm by Eshelman [88]. His experiments have shown that this approach performs

Table 5.5: Performance of six algorithms on 6 *NK*-landscapes

Alg.	**gen**	**Fitness**	**gen**	**Fitness**	**gen**	**Fitness**
	$K = 2, N = 512$		$K = 4, N = 512$		$K = 11, N = 512$	
GA-UX	31569	0.735041, 1.58%	29356	0.762612, 3.54%	21179	0.735579, 4.44%
GA-1X	42978	0.722648, 3.24%	39047	0.745724, 5.67%	25770	0.720248, 6.43%
GA-M	66270	0.730120, 2.24%	58505	0.762168, 3.59%	35516	0.743602, 3.39%
MA-C	20902	0.746324, 0.07%	8585	0.787023, 0.45%	269	0.744263, 3.31%
MA-M	25640	0.745696, 0.15%	12947	0.778917, 1.47%	2907	0.763278, 0.84%
MLS	9183	0.727272, 2.62%	5366	0.752791, 4.78%	2376	0.739058, 3.99%
Best		0.746840, 0.00%		0.790573, 0.00%		0.769733, 0.00%
Time:						600 s
	$K = 2, N = 1024$		$K = 4, N = 1024$		$K = 11, N = 1024$	
GA-UX	31183	0.735720, 1.85%	29000	0.753166, 4.23%	20913	0.730662, 3.89%
GA-1X	42214	0.721064, 3.80%	38173	0.734302, 6.63%	25293	0.708768, 6.77%
GA-M	68301	0.731607, 2.40%	60088	0.752943, 4.26%	36581	0.737865, 2.94%
MA-C	12615	0.748230, 0.18%	4540	0.783665, 0.35%	105	0.732874, 3.60%
MA-M	21486	0.747369, 0.29%	10915	0.775932, 1.34%	2522	0.756074, 0.55%
MLS	3669	0.723803, 3.44%	2344	0.743459, 5.46%	1014	0.731868, 3.73%
Best		0.749580, 0.00%		0.786437, 0.00%		0.760222, 0.00%
Time:						1200 s

significantly better than the standard genetic algorithm. The fourth and fifth algorithm (MA-C and MA-M) are memetic algorithms using *1-opt* local search. In MA-C, the HUX recombination operator is used to create offspring while the MA-M performs flip mutation: three bits are flipped simultaneously in the genome. In order to circumvent the problem of premature convergence, restarts are performed when the population has converged (i.e. when the average Hamming distance between individuals of the population has become smaller than a threshold). To restart the search, all members of the population except the best one are mutated by flipping k randomly chosen bits; k is determined by a third of the average Hamming distance between the individuals in the initial population. This value for k exhibited good performance in several experiments.

The sixth and last algorithm (MLS) in the first set of experiments is a simple multi-start local search: Independent randomly generated solutions are improved by a local search algorithm and the best solution obtained is returned.

The evolutionary algorithms described above were tested on 15 landscapes. Test set A contains five landscapes with $K = 2$ and $N \in \{12, 64, 256, 512, 1024\}$, set B contains five landscapes with $K = 4$ and $N \in \{12, 64, 256, 512, 1024\}$, and set C contains five landscapes with $K = 11$ and $N \in \{12, 64, 256, 512, 1024\}$. All landscapes are based on a random neighbor model. Test set A and B consist of problems with low epistasis and test set C consists of problems with relatively high epistasis. Higher values of K could not be chosen due to the high amount of memory required to store the fitness contribution tables, since the memory requirements grow with $O(N\ 2^{K+1})$. Each of the six algorithms was applied to the 15 test problems of different K and N. In order to enable a fair comparison, the running time of the algorithms was fixed: after a predefined time limit the algorithms were terminated, and the best solution found so far was recorded. The population size for GA-

UX, GA-1X and GA-M was set to 100, and the memetic algorithms (MA-C and MA-M) operated on a population of only 20 members due to the relatively long computation times needed to produce the local optima. Tables 5.4 and 5.5 summarize the results. The first column denotes the algorithm used; the entry "Best" shows the highest fitness ever found by one of the algorithms. The second, fourth and sixth column contain the value for the average number of generations evolved. The third, fifth and seventh column display the the average fitness values obtained over twenty runs. Furthermore, the percentage excess over the best solution found is given in parentheses. The time limit chosen per run is given in seconds on a PC (Pentium II, 300 MHz) under Linux; all algorithms were programmed in *C++*.

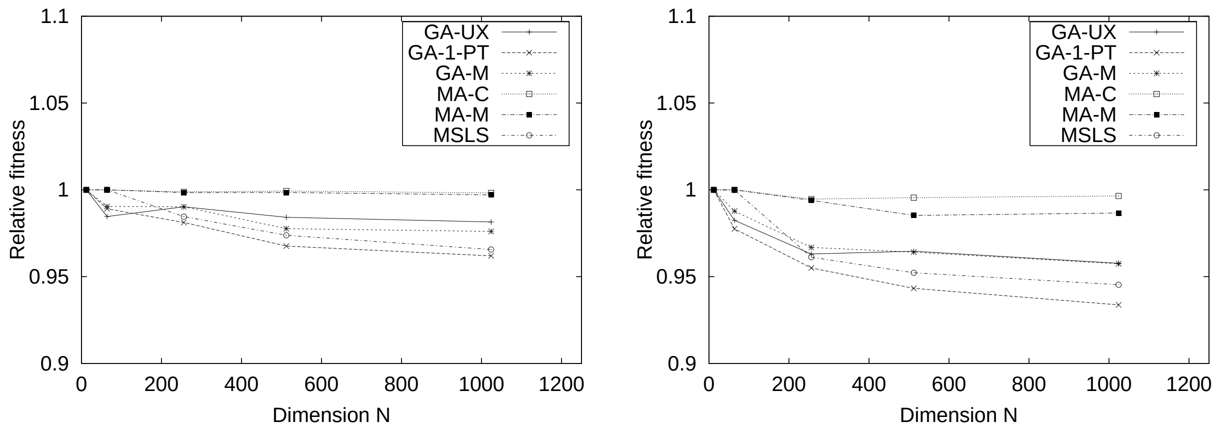


Figure 5.4: Relative fitness over N for $K = 2$ and $K = 4$

Figure 5.4 shows the relative fitness of the algorithms for $K = 2$ and $K = 4$ graphically, i.e. the best objective value ever found divided by the average objective value obtained by the specific algorithm over twenty runs. Figure 5.5 displays the relative fitness of the algorithms for $K = 11$. The results show that for the lowest dimension ($N = 12$) all algorithms perform

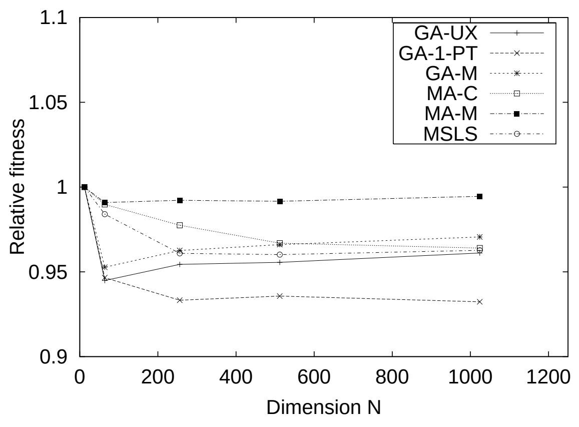


Figure 5.5: Relative fitness over N for K=11

equally good. For $N = 64$ the genetic algorithms (GA-*) are outperformed by multi–start local search (MLS).

Surprisingly, GA-M performs better than GA-UX for $K = 2$ and $N = 64$. For $K = 4$, this is true even for $N = 256$. For greater N, GA-UX performs slightly better than the mutation–only evolutionary algorithm. In case of $K = 11$, GA-M is better than the crossover–based GAs for all $N > 12$.

GA-1X however performs worst in all test cases. It appears that one–point crossover is not a good recombination strategy if the interacting genes for each locus are selected randomly among the N, as it is the case in the test instances. Here, favorable combinations of genes - the building blocks - are not made up by sequences of consecutive genes in the genome. Additional experiments were performed by generating landscapes with the adjacent neighborhood model. The experiments conducted support the claim made: here GA-1X performed much better.

The memetic algorithms on the other hand performed better than the simple EAs in all cases. The relative difference in performance increases significantly with the dimension of the search space. Again, for $K = 11$, the mutation based search is preferable to the crossover based search: MA-M outperforms MA-C by far. One reason may be the fact that the average number of generations evolved by MA-C is much less than the average number of generations for MA-M (105 vs. 2522) because the time consumed by the local search procedure is much higher in case of MA-C. For $K = 2$ and $K = 4$ MA-C beats MA-M.

Figure 5.4 through Figure 5.5 give hints on the relative performance when the problem size increases. The GA-* algorithms as well as the MLS algorithm do not scale well compared to the MAs. Assuming that the MAs' ability to reach the optimum solution also decreases with the problem size, the efficiency of the GA-* algorithms degrades even faster than displayed in the figures.

Greedy and Local Search

To investigate the relative performance of the greedy heuristic and the *k-opt* local search, experiments were conducted in which the two together with the *1-opt* local search were applied to the three landscapes with $N = 1024$ used in the analysis and in the comparison above. The results are shown in Table 5.6. In the table, the average performance (fitness and aver-

Table 5.6: Performance of the Greedy Heuristic, *1-opt* and *k-opt* Local Search

	Greedy		**1-opt LS**		**k-opt LS**	
Instance	fitness	t/ms	fitness	t/ms	fitness	t/ms
C2-1024	0.732613 (2.33%)	76.9	0.713496 (4.88%)	19.7	0.725135 (3.32%)	52.3
D4-1024	0.756306 (4.34%)	173.2	0.723684 (8.47%)	28.4	0.751549 (4.94%)	114.8
B11-1024	0.726171 (5.56%)	22120	0.709401 (7.74%)	112.5	0.740319 (3.72%)	677.3

age percentage access in parentheses) and the average running time (t/ms) in milliseconds of a single run, is shown for the greedy heuristic and *1-opt* and *k-opt* local search applied to randomly generated solutions. The values are averaged over 10000 runs except for the greedy heuristic and the problem instance B11-1024: Due to the long running time, 1000 runs were performed instead of 10000. The values given in parentheses denote the deviation from the best known solution in percent. Note, that these values are not comparable with the values presented in Table 5.5, since in the later, the percentage access is provided relative to the best of the runs in the comparison. The best-known solution has been found in a long run of a MA algorithm with *k-opt* local search.

For $K = 2$ and $K = 4$, the greedy heuristic outperforms the local searches but requires more CPU time. For $K = 11$, the *k-opt* local search dominates over the two others. The CPU time required for a run of the greedy algorithm grows to 22 seconds and is thus more than 32 times higher than for *k-opt* local search rendering the greedy heuristic impractical for such relative large K. The greedy heuristic is furthermore capable of producing comparable results in a single run and thus in 173 milliseconds to a GA/EA requiring 1200 seconds for $K = 2$. For $K = 4$ and $K = 11$, the GAs/EAs are outperformed by the greedy heuristic and the *k-opt* local search in a single run, demonstrating even more drastically the inferior performance of traditional GAs/EAs on relatively large instances.

Memetic Algorithms with *k-opt* Local Search

To assess the performance of memetic algorithms with *k-opt*, additional experiments have been conducted. With the same time limit as chosen for the comparison of evolutionary algorithms with MAs, the MAs with *k-opt* local search were applied to the three instances of size 1024. With a population size of 40, the production of 20 new offspring per generation, and restarts enabled, the MA were run with three different variation operators. The first MA uses the greedy heuristic in the initialization phase and the greedy recombination operator (GX). The second MA uses HUX as the recombination operator and the third MA uses the mutation operator like the MA denoted MA-M in the comparison above. The results of the experiments are summarized in Table 5.7. For each algorithm, the average number of

Table 5.7: Performance of *k-opt* Local Search MAs on three *NK*-landscapes

	C2-1024		D4-1024		B11-1024	
Op	gen	fitness, quality	gen	fitness, quality	gen	fitness, quality
GX	12505	0.75000189, 0.01%	5750	0.78757009, 0.39%	3	0.75677286, 1.58%
HUX	11954	0.75000956, 0.01%	5730	0.78687418, 0.48%	216	0.75356503, 1.99%
MUT	6402	0.74475693, 0.71%	4306	0.77277608, 2.26%	704	0.75574657, 1.71%
Best		0.75006522, 0.00%		0.79064095, 0.00%		0.768882, 0.00%

generations (gen) produced is provided as well as the average fitness (fitness) of the final solution along with the percentage access over the best known solution (quality).

For $K = 2$, the MA with greedy recombination and HUX recombination perform equally well. Both find the best known solution in one out of 20 runs and have the same worst result. For $K = 4$ and $K = 11$, the greedy recombination MA outperforms the others. The mutation based MA is as expected the worst out of the three for $K = 2$ and $K = 4$. For $K = 11$, the mutation based MA achieves a better average result than the MA with HUX, but its performance is still below the performance of the greedy recombination MA after the third generation. However, mutation is still preferable over recombination for the $K = 11$ landscape since the good result of the greedy recombination MA stems from the initialization of the greedy heuristic rather than from the effectiveness of recombination. The recombination base MAs with *k-opt* local search perform clearly better than the algorithms with *1-opt* local search. Furthermore, these algorithms have a higher potential and perform better if longer running times are chosen.

5.5 Summary

NK-landscapes have been introduced as a formal model of gene interaction in biological evolution, and since they are random, several statistical properties of the landscapes are known. In this chapter, the characteristics of *NK*-landscapes have been summarized including their autocorrelation and random-walk correlation functions, their fitness distance correlation, and their computational complexity. Furthermore, to derive highly effective memetic algorithms for the *NK*-model, two new heuristics have been proposed, a greedy algorithm and a *k-opt* local search. The distribution of the solutions produced by these heuristics has been analyzed by performing a fitness distance correlation analysis on selected instances. The results allow to predict when greedy choices based on the greedy heuristic are favorable in a memetic framework and when not. Additionally, investigating the distribution of local optima in the landscapes allows to determine whether or not recombination is effective.

To assess the performance of the memetic approach, MAs have been compared with traditional evolutionary algorithms with focus on genetic algorithms that do *not* incorporate heuristic components. The *NK*-model offers the best conditions for the application of GAs for two reasons. First, *NK*-landscapes are binary-coded problems with no explicit or implicit constraints: each binary vector of length N represents a feasible solution. In contrast to other combinatorial optimization problems such as the traveling salesman problem or the knapsack problem, the traditional crossover operators of genetic algorithms can be applied without modification to the *NK*-model. Second, the *NK*-model provides instances with low epistasis for which GAs are claimed to be best suited [68, 67].

Nevertheless, memetic algorithms using simple local search have been shown to scale much better with the problem size; traditional EAs/GAs quickly become ineffective with increasing problem size. Furthermore, it has been shown that the more sophisticated greedy heuristic produces in a single run and thus in a few milliseconds solutions that are better than those found by a GA in 20 minutes on landscapes with low epistasis ($K = 4$). The greedy heuristic incorporated in the initialization phase as well as in the recombination operator of a MA with *k-opt* local search is shown to be highly effective for landscapes with low epistasis. The landscape analysis has shown that with increasing epistasis, the landscape becomes rapidly unstructured. Thus, for these instances, a *k-opt* local search MA with mutation instead of recombination has been shown to be favorable.

Chapter 6

The Binary Quadratic Programming Problem

6.1 Introduction

The unconstrained binary quadratic programming problem (BQP) of the form $f(x) = x^t Q x$, $x \in \{0,1\}^n$ (Q is a $n \times n$ matrix) is an important combinatorial optimization problem. Several other combinatorial problems can be reformulated as a binary quadratic programms, as for example, the maximum clique problem. Moreover, it allows for the straight-forward application of genetic algorithms, since a solution to the problem is a binary vector of fixed length. Therefore, the BQP is of special interest not only from the viewpoint of problem transformation but also in heuristic algorithm design.

In this chapter, new greedy and local search heuristics for the BQP are proposed. In particular, a *k-opt* local search is devised based on ideas used in the Lin-Kernighan heuristic for the traveling salesman problem and the Kernighan-Lin heuristic for graph partitioning. Moreover, a fitness landscape analysis of commonly used benchmark instances is performed to identify problem characteristics influencing the performance of memetic algorithms. Compared to well-known landscape properties of *NK*-landscapes, landscape ruggedness does not increase with increasing epistasis for the studied BQP instances. Furthermore, it is shown that the local optima are found in a small fraction of the search space and that there is a high correlation between the objective value of local optima and their hamming distance to the optimum or best-known solution. In particular, the local optima found by the *k-opt* local search are shown to be very similar.

The newly proposed greedy and local search heuristics are evaluated in experiments on 105 problem instances. It turns out that a randomized variant of the *k-opt* local search is highly effective. Additionally, a comparison of genetic algorithms and memetic algorithms is performed. The results indicate that a memetic algorithm incorporating a simple *1-opt* local search scales much better with the problem size than simple GAs. For large problem instances, a memetic algorithm employing the greedy heuristic in the initialization phase and the randomized *k-opt* local search for producing local optima is proposed, and its effectiveness is shown on the largest instances available to the author. Due to the similarity of the local optima of the studied instances, a mutation-based MA with a mutation rate derived from the landscape analysis is proven to be superior to a recombination-based MA.

Some of the results presented in this chapter are published in [215, 218].

6.2 Heuristics for the BQP

Several exact methods have been developed to solve the BQP [244, 19, 32, 133], but due to the computational complexity of the problem, heuristics have been proposed recently to find solutions to large problem instances, including *tabu search* [26, 116, 115], *scatter search* [6], *simulated annealing* [26, 166] and *evolutionary algorithms* [194, 215].

Since greedy heuristics and local search are well suited for the incorporation into an evolutionary framework, new algorithms of both types of heuristics are presented in the following paragraphs. It is assumed that the BQP is defined as maximizing

$$f(x) = x^t\,Q\,x = \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}\,x_i\,x_j, \tag{6.1}$$

with $x \in \{0,1\}^n$ and $q_{ij} \in I\!R$.

6.2.1 Greedy Heuristics

Greedy algorithms are intuitive heuristics in which greedy choices are made to achieve a certain goal. In combinatorial optimization, a solution to a given problem is searched which maximizes or minimizes an objective function. Greedy heuristics are constructive heuristics since they construct feasible solutions for optimization problems from scratch by making the most favorable choice in each step of construction. By adding an element to the (partial) solution which promises to deliver the highest gain, the heuristic acts as a “greedy constructor”.

A Simple Greedy Algorithm

In the BQP, a solution is constructed by assigning a binary value to an element in the solution vector x: a zero or a one. The general outline of the greedy algorithm is provided in Figure 6.1. In each step, the heuristic searches for an element k in the solution vector and

```
**procedure** Greedy($x \in X$): $X$;
  **begin**
    $C := \{1,\ldots,n\}$;
    **repeat**
      find $k, l$ with $g_k^l = \max_{i \in C,\, j \in \{0,1\}} g_i^j$;
      $x_k := l$;
      $C := C \backslash \{k\}$;
    **until** $C = \emptyset$;
    **return** $x$;
  **end;**
```

Figure 6.1: A Greedy Heuristic for the BQP

a value l to assign to it so that a gain function g_k^l is maximized. Afterwards, the value l is assigned to the vector component x_k.

To find an appropriate gain function, we modify the problem by adding a third state: Let $y \in Y = \{0, \frac{1}{2}, 1\}^n$ be a vector in which each component y_i can have three values 0, $\frac{1}{2}$, and 1. Starting with a vector $\hat{y}$ with $\hat{y}_i = \frac{1}{2}$ for all i, the greedy heuristic can be viewed as a transformation algorithm that transforms $\hat{y}$ into a vector x for which $x_i \in \{0, 1\}$ for all i and thus $x \in X$. The objective of the solution $\hat{y}$ is

$$f(\hat{y}) = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij}\, \hat{y}_i\, \hat{y}_j = \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij}. \tag{6.2}$$

Let $\tilde{y} \in Y$ be a vector that is equal to $y \in Y$ except for the component y_k. Then

$$\Delta f(y) = f(\tilde{y}) - f(y) = q_{kk}\,(\tilde{y}_k^2 - y_k^2) + 2\,(\tilde{y}_k - y_k) \sum_{j=1, j\neq k}^{n} q_{kj}\, x_j. \tag{6.3}$$

Hence, the gain for changing y_k from 0.5 to 0 (denoted g_k^0) or 1 (denoted g_k^1) can be defined as

$$g_k^1 = \frac{3}{4} q_{kk} + \sum_{j=1, j\neq k}^{n} q_{kj}\, y_j, \qquad \text{and} \qquad g_k^0 = -\frac{1}{4} q_{kk} - \sum_{j=1, j\neq k}^{n} q_{kj}\, y_j. \tag{6.4}$$

Using this formula, the greedy heuristic as displayed in Figure 6.1 has a runtime complexity of $O(n^3)$ since the calculation of all g_i takes $O(n^2)$ and this has to be done n times before the algorithm terminates. Thus, the greedy heuristic has an expected running time equal to an algorithm that calculates the objective function $f(x)$ for n solutions. However, the greedy algorithm can be implemented more efficiently. If the matrix Q is not stored in a two-dimensional array, but instead, for each i in $\{1, \ldots, n\}$ a list of j's for which $q_{ij} \neq 0$ is maintained, the running time is reduced considerably for instances with a low density. To reduce the running time to find a choice with a highest gain, the following formula can be used to calculate the new gain g_i after y_k has been set to 0 or 1:

$$\Delta g_i^1 = \begin{cases} \frac{1}{2}\, q_{ik} & \text{if} \quad y_k = 1 \\ -\frac{1}{2}\, q_{ik} & \text{otherwise} \end{cases} \qquad \text{and} \qquad \Delta g_i^0 = \begin{cases} -\frac{1}{2}\, q_{ik} & \text{if} \quad y_k = 1 \\ \frac{1}{2}\, q_{ik} & \text{otherwise} \end{cases} \tag{6.5}$$

Thus, once the gains g_i have been calculated, they can be efficiently updated after the component y_k has been set. Since the gains g only change for those i with $q_{ik} \neq 0$, the running time to update the gains has a complexity of $O(n)$. In Figure 6.2, the pseudo code of the fast greedy heuristic is provided. The running time for the improved heuristic is in $O(n^2)$ since the running time to find the maximum gain g_k^0 and g_k^1 is in $O(n)$ and has to be performed n times to construct a feasible solution. Thus, the greedy heuristic has the same computational complexity as an algorithm calculating $f(x)$ for a single solution. The running time behavior of the greedy heuristic can be improved even further if additional data structures are used to find the maximum gain in shorter than $O(n)$ time. However, this has no influence on the computational complexity of the algorithm since the time for the initial calculation of the gains dominates over the times for the greedy choices.

A Randomized Greedy Algorithm

The greedy heuristic described above is deterministic, since it always produces the same solution for a given problem instance. Often, it is desired that a construction heuristic

```
procedure Greedy($x \in X$): $X$;
  begin
    $C := \{1, \dots, n\}$;
    for $i := 1$ to $n$ do $x_i = \frac{1}{2}$;
    calculate gains $g_i$ for all $i$ in $\{1, \dots, n\}$;
    repeat
      find $k_0$ with $g_k^0 = \max_{i \in C} g_i^0$;
      find $k_1$ with $g_k^1 = \max_{i \in C} g_i^1$;
      if $g_{k_0}^0 > g_{k_1}^1$ then
        $x_{k_0} := 0$; $C := C \backslash \{k_0\}$;
      else
        $x_{k_1} := 1$ $C := C \backslash \{k_1\}$;
      endif
      update gains $g_i$ for all $i \in C$;
    until $C = \emptyset$;
    return $x$;
  end;
```

Figure 6.2: A Fast Greedy Heuristic for the BQP

produces many different solutions, for example in hybrid algorithms. The above procedure can be randomized as follows:

By making the first choice randomly, i.e., selecting k and l randomly and setting $x_k = l$ in the first step, a random component is incorporated. Furthermore, the deterministic choice among x_{k_0} and x_{k_1} can be replaced by a random choice proportional to the gains $g_{k_0}^0$ and $g_{k_1}^1$: x_{k_0} is set to 0 with probability $\frac{g_{k_0}^0}{g_{k_0}^0 + g_{k_1}^1}$ and x_{k_1} is set to 1 with probability $\frac{g_{k_1}^1}{g_{k_0}^0 + g_{k_1}^1}$. The pseudo code of the randomized greedy heuristic is provided in Figure 6.3.

6.2.2 Local Search

Local search (LS) algorithms are improvement heuristics that search in the neighborhood of the current solution for a better one until no further improvement can be made, i.e. there is no better solution in the neighborhood of the current solution. Local search algorithms can be categorized by the neighborhoods they consider. For example, the neighborhood of a solution represented by a binary vector can be defined by the solutions that can be obtained by flipping a single or multiple components in the binary vector simultaneously.

1-opt Local Search

The simplest form of local search for the BQP is the *1-opt* local search: In each step, a new solution with a higher fitness in the neighborhood of the current solution is searched. The neighborhood of the current solution is defined by the set of solutions that can be reached by flipping a single bit. Hence, the *1-opt* neighborhood contains all solutions with a hamming distance of 1 to the current solution. In our implementation, we search for the solution

```
procedure RandomizedGreedy($x \in X$): $X$;
  begin
    $C := \{1, \dots, n\}$;
    for $i := 1$ to $n$ do $x_i = \frac{1}{2}$;
    calculate gains $g_i$ for all $i$ in $\{1, \dots, n\}$;
    Select $k, l$ randomly and set $x_k := l$;
    $C := C \backslash \{k\}$;
    repeat
      find $k_0$ with $g_k^0 = \max_{i \in C} g_i^0$;
      find $k_1$ with $g_k^1 = \max_{i \in C} g_i^1$;
      set $p = \frac{g_{k_0}^0}{g_{k_0}^0 + g_{k_1}^1}$;
      if randomNumber[0,1]$< p$ then
        $x_{k_0} := 0$; $C := C \backslash \{k_0\}$;
      else
        $x_{k_1} := 1$; $C := C \backslash \{k_1\}$;
      endif
      update gains $g_i$ for all $i \in C$;
    until $C = \emptyset$;
    return $x$;
  end;
```

Figure 6.3: A Randomized Greedy Heuristic for the BQP

with the highest fitness, i.e. we search for a flip with the highest associated gain in fitness ($g = f_{new} - f_{old}$). The gain g_k of flipping bit k in the current solution can be calculated in linear time using the formula

$$g_k = q_{kk}(\overline{x}_k - x_k) + 2 \sum_{i=1, i \neq k}^{n} q_{ik}\, x_i\, (\overline{x}_k - x_k), \tag{6.6}$$

with $\overline{x}_k = 1 - x_k$.

The local search algorithm is given in pseudo code in Figure 6.4. A straightforward implementation of the *1-opt* local search displayed in the figure has a running time of $O(n^2)$ per iteration. Analogous to the greedy heuristic, the efficiency of the algorithm can be increased considerably. The gains g_i do not have to be recalculated each time. Instead, it is sufficient to calculate the difference of the gains Δg_i. Assuming that all g_i for a BQP solution have been calculated and the bit k is flipped, the new gains g_i' can be calculated efficiently by the formula:

$$g_i' = \begin{cases} -g_i & \text{if} \quad i = k \\ g_i + \Delta g_i(k) & \text{otherwise} \end{cases} \qquad \text{with} \qquad \Delta g_i(k) = 2\, q_{ik}\, (\overline{x}_i - x_i)\, (x_k - \overline{x}_k) \tag{6.7}$$

Thus, the update of the gains can be performed in linear time. Furthermore, only the gains g_i for $q_{ik} \neq 0$ have to be updated. The fast *1-opt* local search is displayed in Figure 6.5. The running time of this algorithm is $O(n)$ per iteration. The initialization of the gains is

```
procedure Local-Search-1-opt($x \in X$): $X$;
   begin
      repeat
         find $k$ with $g_k = \max_i g_i$;
         if $g_k > 0$ then $x_k := 1 - x_k$;
      until $g_k \leq 0$;
      return $x$;
   end;
```

Figure 6.4: *1-opt* Local Search

```
procedure Local-Search-1-opt($x \in X$): $X$;
   begin
      calculate gains $g_i$ for all $i$ in $\{1, \ldots, n\}$;
      repeat
         find $k$ with $g_k = \max_i g_i$;
         if $g_k > 0$ then
            $x_k := 1 - x_k$;
            update gains $g_i$;
         endif
      until $g_k \leq 0$;
      return $x$;
   end;
```

Figure 6.5: Fast 1–opt Local Search for the BQP

in $O(n^2)$.

k-opt Local Search

The k-opt neighborhood $\mathcal{N}_{k\text{-opt}}$ of a binary vector of length n is defined by the binary vectors that can be reached by flipping one up to k bits in the vector simultaneously. Hence, the neighborhood $\mathcal{N}_{k\text{-opt}}(x) = \{x' \in X | d_H(x, x') \leq k\}$ (d_H denotes the hamming distance between bit vectors) grows exponentially with k: $|\mathcal{N}_{k\text{-opt}}| = n^k$.

Since it is computationally too expensive to search the complete *k-opt* neighborhood, Lin and Kernighan have proposed heuristics for the traveling salesman problem (TSP) and the graph partitioning problem that efficiently search a small fraction of the *k-opt* neighborhood. These algorithms, known as the Lin-Kernighan algorithm for the TSP [192], and the Kernighan-Lin algorithm for graph partitioning [171], belong to the best available heuristics for these two combinatorial optimization problems. In the following, a local search algorithm for the BQP is presented that is based on the ideas of Lin and Kernighan.

The basic idea of the heuristic is to find a solution by flipping a variable number of k bits in the solution vector per iteration. In each step, a sequence of n solutions is generated by flipping the bit with the highest associated gain. Analogous to the *1-opt* local search procedure, a vector of gains is maintained and updated according to equation (6.7) after each flip. Furthermore, a candidate set is used to assure that each bit is flipped exactly once. The best solution in the sequence is accepted as the new solution for the next iteration. Thus, in each iteration of the algorithm a variable number of bits are flipped to find a new solution in the neighborhood of the current solution. The pseudo code of the k-opt local search is presented in Figure 6.6.

```
procedure Local-Search-k-opt($x \in X$): $X$;
  begin
    calculate gains $g_i$ for all $i$ in $\{1, \dots, n\}$;
    repeat
      $x_{prev} := x$, $G_{max} := 0$, $G := 0$, $C := \{1, \dots, n\}$;
      repeat
        find $j$ with $g_j = \max_{i \in C} g_i$;
        $G := G + g_j$;
        if $G > G_{max}$ then $G_{max} := G$, $x_{best} := x$;
        $x_j := 1 - x_j$, $C := C \backslash \{j\}$;
        update gains $g_i$ for all $i$;
      until $C = \emptyset$;
      if $G_{max} > 0$ then $x := x_{best}$ else $x := x_{prev}$;
    until $G_{max} \leq 0$;
    return $x$;
  end;
```

Figure 6.6: *k-opt* Local Search for the BQP

The runtime complexity for the initialization is in $O(n^2)$ and the running time per iteration is also of complexity $O(n^2)$.

The reduce the running time, the termination condition of the inner repeat-loop can be modified so that the loop is terminated if there were no new x_{best} for more than m iterations. Thus, the resulting *fast k-opt* procedure has a shorter running time for $m \ll n$ than the *k-opt* procedure described before.

A Randomized k-opt Local Search

When a local search is used in meta-heuristics, it is often desired that the probability to fall in a previously discovered local optimum is low. One way to achieve this is to randomize the local search. Katayama [164, 167] has proposed the following randomization scheme for the k-opt local search introduced above. In the inner loop, before the bits are flipped in decreasing order of associated gain, the bits are considered in random order and they are flipped if the associated gain is positive. The outline of the algorithm is provided in Figure 6.7; the added lines are marked with an asterix. The randomization is essentially a randomized fast *1-opt* local search with a fixed number of iterations ($\leq n$). It will be shown later that this combination is very effective for the BQP.

```
procedure Randomized-Local-Search-k-opt($x \in X$): $X$;
  begin
    calculate gains $g_i$ for all $i$ in $\{1,\dots,n\}$;
    repeat
      $x_{prev} := x$, $G_{max} := 0$, $G := 0$, $C := \{1,\dots,n\}$;
      repeat
*       Generate a random permutation RP[] of the set $\{1,\dots,n\}$;
*       for $j := 1$ to $n$ do
*         k = RP[j];
*         if $g_k > 0$ then
*           $G = G + g_k$, $G_{max} = G$;
*           $x_j := 1 - x_j$, $x_{best} := x$;
*           update gains $g_i$ for all $i$;
*           $C := C \setminus \{j\}$;
*         endif
*       endfor
        find $j$ with $g_j = \max_{i \in C} g_i$;
        $G := G + g_j$;
        if $G > G_{max}$ then $G_{max} := G$, $x_{best} := x$;
        $x_j := 1 - x_j$, $C := C \setminus \{j\}$;
        update gains $g_i$ for all $i$;
      until $C = \emptyset$;
      if $G_{max} > 0$ then $x := x_{best}$ else $x := x_{prev}$;
    until $G_{max} \le 0$;
    return $x$;
  end;
```

Figure 6.7: Randomized *k-opt* Local Search for the BQP

6.3 The Fitness Landscape of the BQP

Since the local search procedures introduced above rely on the k-opt neighborhood as defined as

$$\mathcal{N}_{k\text{-}opt}(x) = \{x' \in X | d_H(x', x) \le k\} \tag{6.8}$$

where d_H denotes the hamming distance between bit strings and X the set of all bit strings of length n ($X = \{0,1\}^n$), the landscape considered in the search space analysis of the BQP is $\mathcal{L} = (X, f, d_H)$. The graph of this landscape is a hypercube of dimension n in which the nodes represent the (candidate) solutions of the problem. An edge in the graph connects neighboring points in the landscape, i.e. points that have hamming distance 1.

6.3.1 Epistasis in the BQP

There is a close relation between binary quadratic programming and NK-landscapes as defined by Kauffman [169]. The objective function of the BQP can be decomposed into n

functions. The fitness of a BQP solution can thus be rewritten as a sum of functions for each site, called the fitness contributions f_i of site i in the genome:

$$f(x) = \sum_{i=1}^{n} f_i(x_i, x_{i_1}, \ldots, x_{i_{k(i)}}), \tag{6.9}$$

$$f_i(x) = \sum_{j=1}^{n} q_{ij}\, x_i\, x_j \tag{6.10}$$

Similar to the NK-landscapes defined in [169], the fitness contribution f_i of a site i depends on the gene value x_i and of $k(i)$ other genes $x_{i_1}, \ldots, x_{i_{k(i)}}$. While for NK-landscapes $k(i) = K$ is constant for all i, in the BQP $k(i)$ is defined as the number of non-zero entries in the i-th column of matrix Q. Hence, the degree of epistasis in a BQP instance can be easily determined by calculating the density of the matrix Q. It is defined as the number of non-zero entries divided by the number of total entries in the matrix. Thus, the density is between 0 and 1, where 0 means no epistasis and 1 maximum epistasis (every gene depends on the values of all other genes).

The density of an instance and hence epistasis in the problem is expected to have an influence on the fitness landscape and thus on the hardness of A BQP instance for heuristics as, observed in other problems like *NK*-landscapes.

6.3.2 Autocorrelation Analysis

Since there are no theoretical results on the autocorrelation function or the random walk correlation function for the BQP, experiments have been conducted to estimate the correlation length of selected landscapes. In Table 6.1, the instances selected for the analysis are listed along with their size n and their matrix density $dens(Q)$. The instances denoted glov500 and beas2500 have been introduced in [116] and [26], respectively; both sets are contained in ORLIB [25]. The instances with prefix kb-g have been provided by G. Kochenberger and are used in the performance evaluation of tabu search and scatter search [115, 6]. The results of the random walk correlation analysis are displayed in the last two columns of Table 6.1: The normalized correlation length (n/ℓ) and the correlation length itself (ℓ) are provided.

Considering all selected instances, the quotient of n/ℓ varies in tight bounds: the lowest value for n/ℓ is 2.36 and the highest is 2.71. Compared to *NK*-landscapes, this is fairly low since in the *NK*-model $n/\ell \approx K + 1$. For the instances denoted glov500, the values are very similar (2.67 ± 0.04) and thus remain constant independent of the density of the problem. For the set kb-g, the values for n/ℓ do change with the density of Q, but the values become smaller with increasing density. This is surprising, since in the *NK*-model, the correlation length decreases with increasing epistasis, and the density can be regarded as a measure of epistasis in the BQP. For the set of instances of size 2500 and a density of 0.1, the values for n/ℓ are constant (about 2.66).

Summarizing, all the instances of the BQP considered here have got a smooth landscape similar to *NK*-landscapes with $K \leq 3$.

6.3.3 Fitness Distance Correlation Analysis

In a second analysis, the correlation of fitness (objective f(x)) and distance to the optimum was investigated for local optima with respect to *1-opt* and *k-opt* local search. The results

Table 6.1: Results of the Fitness Distance Analysis for 1-opt Solutions of the BQP

Instance	n	$dens(Q)$	$\min d_{opt}$	$\overline{d}_{opt}$	$\overline{d}_{loc}$	$N_{1\text{-}opt}$	ϱ	n/ℓ	ℓ
glov500-1	500	0.1	11	68.58 (0.14)	80.14	2500	-0.79	2.64	189.30
glov500-2	500	0.2	0	68.29 (0.14)	80.03	2498	-0.74	2.64	189.63
glov500-3	500	0.5	16	74.52 (0.15)	86.18	2498	-0.75	2.69	185.94
glov500-4	500	0.7	0	81.36 (0.16)	78.76	2499	-0.31	2.67	187.39
glov500-5	500	1.0	17	76.13 (0.15)	89.01	2498	-0.77	2.71	184.59
kb-g01	1000	0.1	42	160.17 (0.16)	177.31	2500	-0.69	2.61	382.71
kb-g02	1000	0.2	73	177.11 (0.18)	209.76	2500	-0.72	2.55	391.96
kb-g03	1000	0.3	54	213.61 (0.21)	229.71	2500	-0.72	2.55	391.83
kb-g04	1000	0.4	94	212.98 (0.21)	219.69	2500	-0.58	2.48	403.42
kb-g05	1000	0.5	40	149.53 (0.15)	171.29	2500	-0.77	2.45	408.25
kb-g06	1000	0.6	55	225.48 (0.23)	251.53	2500	-0.66	2.45	408.03
kb-g07	1000	0.7	68	223.41 (0.22)	243.12	2500	-0.56	2.43	411.25
kb-g08	1000	0.8	66	197.43 (0.20)	224.89	2500	-0.72	2.38	419.75
kb-g09	1000	0.9	112	220.21 (0.22)	250.97	2500	-0.65	2.38	420.13
kb-g10	1000	1.0	92	229.95 (0.23)	240.97	2500	-0.63	2.36	424.12
beas2500-1	2500	0.1	152	289.12 (0.12)	318.58	2500	-0.74	2.62	953.69
beas2500-2	2500	0.1	131	321.64 (0.13)	335.91	2500	-0.66	2.68	934.12
beas2500-3	2500	0.1	165	298.75 (0.12)	326.03	2500	-0.74	2.67	936.90
beas2500-4	2500	0.1	87	231.85 (0.09)	283.00	2500	-0.82	2.66	941.42
beas2500-5	2500	0.1	102	252.16 (0.10)	291.19	2500	-0.81	2.66	940.87
beas2500-6	2500	0.1	126	264.68 (0.11)	290.24	2500	-0.74	2.66	938.71
beas2500-7	2500	0.1	168	325.28 (0.13)	333.77	2500	-0.53	2.66	940.76
beas2500-8	2500	0.1	84	232.02 (0.09)	270.62	2500	-0.74	2.66	940.30
beas2500-9	2500	0.1	134	259.01 (0.10)	309.77	2500	-0.81	2.65	942.93
beas2500-10	2500	0.1	171	319.61 (0.13)	348.61	2500	-0.75	2.66	940.37

of this analysis for 1-opt local optima are displayed in Table 6.1. In columns four through eight, the minimum distance to the optimum ($\min d_{opt}$), the average distance to the optimum ($\overline{d}_{opt}$) with the average distance divided by n in parentheses, the average distance between the local optima themselves ($\overline{d}_{loc}$), the number of distinct optima out of 2500 ($N_{1\text{-}opt}$), and the fitness distance correlation (FDC) coefficient (ϱ) are provided. Since the global optima are not known for the problems in the analysis, the best-known solutions are used instead which are likely to be the global optima of the instances.

In most cases, the average distance between the local optima and the average distance to the global optimum (best-known solution) are very similar. Moreover, the local optima a located in a small region of the search space: the average distance between the local optima is between a fourth and sixth of the maximum distance (the diameter) between two solutions in the search space. For set glov500, the average distance to the optimum is a sixth of the diameter independent of the density of Q. For set beas2500 the local optima are even closer to the optimum in relation to the maximum distance of two solutions in the landscape: the average distance to other local optima is more than a seventh of the diameter of the landscape. The FDC coefficient varies from -0.56 to -0.81 excluding glov500-4. The FDC coefficient for this instance is -0.31.

In Figure 6.8, some scatter plots are provided in which the distance to the global optimum is plotted against the fitness difference $\Delta f = f(x_{opt}) - f(x_{loc})$ for each local optimum found. As can be seen in the figure, the plot for glov500-4 looks different from all other plots. There

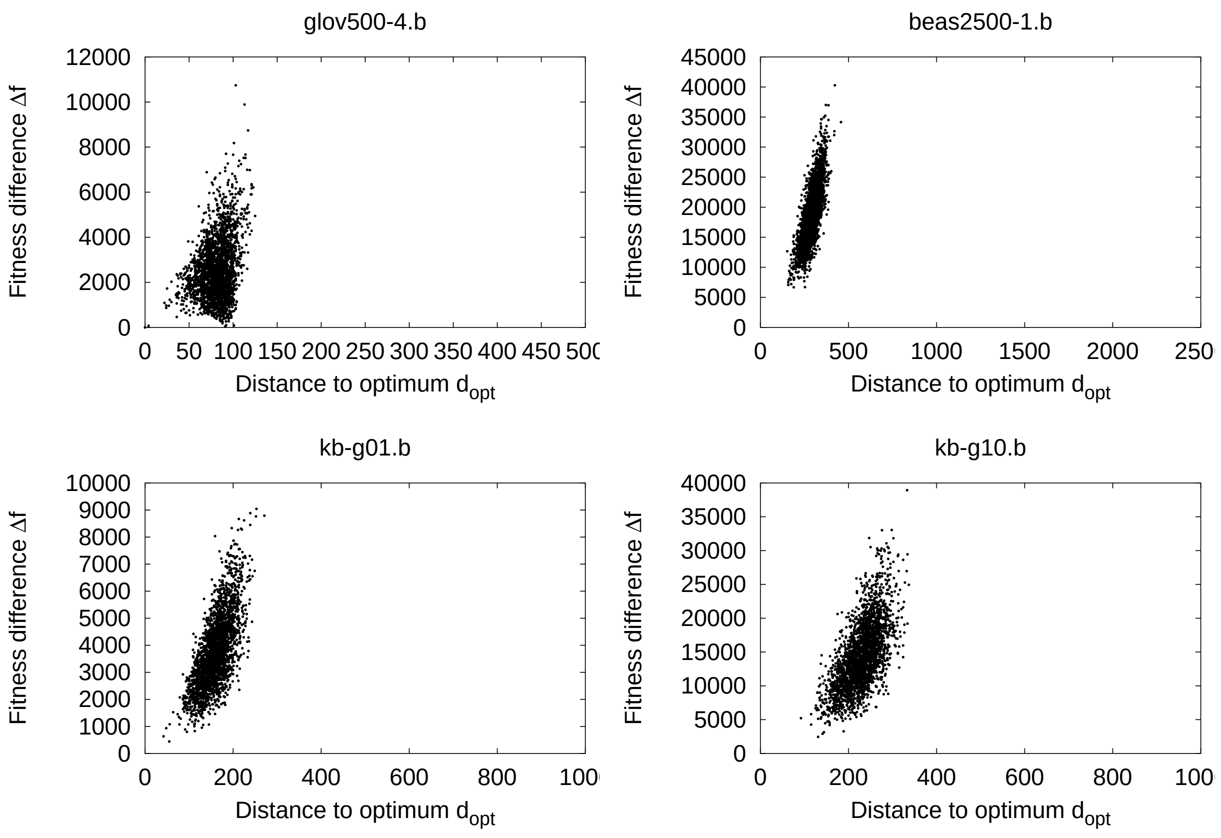


Figure 6.8: Fitness-Distance Plots produced with 1-opt local Search

are solutions with similar fitness to the optimum although they are located at a distance of about 90 units from the optimum. The plots of the other instances in the set glov500 are not displayed since they look like the plot of kb-g01. The instance beas2500-1 is displayed as a representative of set beas2500; the other plots are similar. All plots of set kb-g are similar to the two displayed plots of instance kb-g01 and kb-g10 with only minor differences.

The FDC analysis of *k-opt* local optima produced similar results to the ones described above. In Table 6.2, the results are summarized. Once again, for each instance, the problem size n, the density of Q, the minimum distance to the optimum (min d_{opt}), the average distance to the optimum ($\overline{d}_{opt}$) with the average distance divided by n in parentheses, the average distance between the local optima themselves ($\overline{d}_{loc}$), the number of distinct optima out of 2500 ($N_{k\text{-}opt}$), and the fitness distance correlation coefficient (ϱ) are provided.

The major difference of the properties of *1-opt* and *k-opt* local optima is that the latter are more similar, they are closer to each other in terms of average hamming distance. As a consequence, they are closer to the global optimum (best-known solution). The local optima have more than four fifth of the bit values in common. Averaged over all problems of set beas2500, the average distance to the optimum is about a 12th of the diameter of the landscape. In the set kb-g, the average distance to the optimum lies between a 5th and a 10th of the maximum distance between two points in the search space. Furthermore, it appears that the number of local optima is lower for *k-opt* local search: the number of distinct local optima found is lower than the number of local searches performed for all instances. In many cases, the optimum solution or a solution quite close to the best-known solution could

Table 6.2: Results of the Fitness Distance Analysis for k-opt Solutions of the BQP

Instance	n	$dens(Q)$	$\min d_{opt}$	$\overline{d}_{opt}$	$\overline{d}_{loc}$	$N_{k\text{-}opt}$	ϱ
glov500-1	500	0.1	0	39.68 (0.08)	45.67	1209	-0.77
glov500-2	500	0.2	0	42.46 (0.08)	47.85	1140	-0.79
glov500-3	500	0.5	0	52.36 (0.10)	62.96	1484	-0.74
glov500-4	500	0.7	0	74.91 (0.15)	67.16	1431	0.04
glov500-5	500	1.0	0	50.37 (0.10)	63.63	1421	-0.82
kb-g01	1000	0.1	0	121.11 (0.12)	129.70	2468	-0.59
kb-g02	1000	0.2	0	131.70 (0.13)	154.97	2488	-0.65
kb-g03	1000	0.3	0	150.77 (0.15)	166.36	2415	-0.83
kb-g04	1000	0.4	14	181.94 (0.18)	174.20	2486	-0.47
kb-g05	1000	0.5	0	104.46 (0.10)	111.00	2295	-0.68
kb-g06	1000	0.6	43	182.65 (0.18)	204.64	2499	-0.68
kb-g07	1000	0.7	10	188.65 (0.19)	198.90	2500	-0.52
kb-g08	1000	0.8	0	149.34 (0.15)	156.01	2464	-0.64
kb-g09	1000	0.9	22	184.09 (0.18)	205.82	2500	-0.44
kb-g10	1000	1.0	11	194.29 (0.19)	193.83	2495	-0.54
beas2500-1	2500	0.1	18	220.32 (0.09)	240.11	2500	-0.75
beas2500-2	2500	0.1	95	264.84 (0.11)	255.78	2500	-0.50
beas2500-3	2500	0.1	46	234.34 (0.09)	237.70	2500	-0.52
beas2500-4	2500	0.1	7	158.45 (0.06)	194.17	2499	-0.79
beas2500-5	2500	0.1	16	171.53 (0.07)	202.37	2499	-0.80
beas2500-6	2500	0.1	48	211.23 (0.08)	218.08	2499	-0.56
beas2500-7	2500	0.1	70	280.44 (0.11)	270.35	2500	-0.49
beas2500-8	2500	0.1	0	173.11 (0.07)	189.07	2500	-0.62
beas2500-9	2500	0.1	19	180.21 (0.07)	210.99	2500	-0.71
beas2500-10	2500	0.1	83	246.63 (0.10)	256.75	2500	-0.70

be found in the analysis. However, the fitness distance correlation coefficients are in some cases lower than for *1-opt* local optima. Selected scatter plots are provided in Figure 6.9. Differences of the plots within each set are minor with one exception: As before, glov500-4 shows different properties compared to the four other instances in the set.

In an additional experiment, it has been observed that the randomization of *k-opt* local search above has no significant influence on the results of the analysis.

6.4 A Memetic Algorithm for the BQP

The BQP referred to in this chapter is a binary-coded problem with no additional constraints. Thus, the application of genetic algorithms is a straight-forward task: operators like bit-flip mutation, single-point crossover, two-point crossover, or uniform crossover can be applied without modification. For memetic algorithms, only slight modifications to the operators are required. These modifications are described in the following along with the other details required for the application of evolutionary algorithms to the BQP. The GA and the MA described here are compared experimentally on several BQP instances, afterwards.

6.4.1 A Simple Genetic Algorithm for the BQP

The genetic algorithm for the BQP consists of the following components:

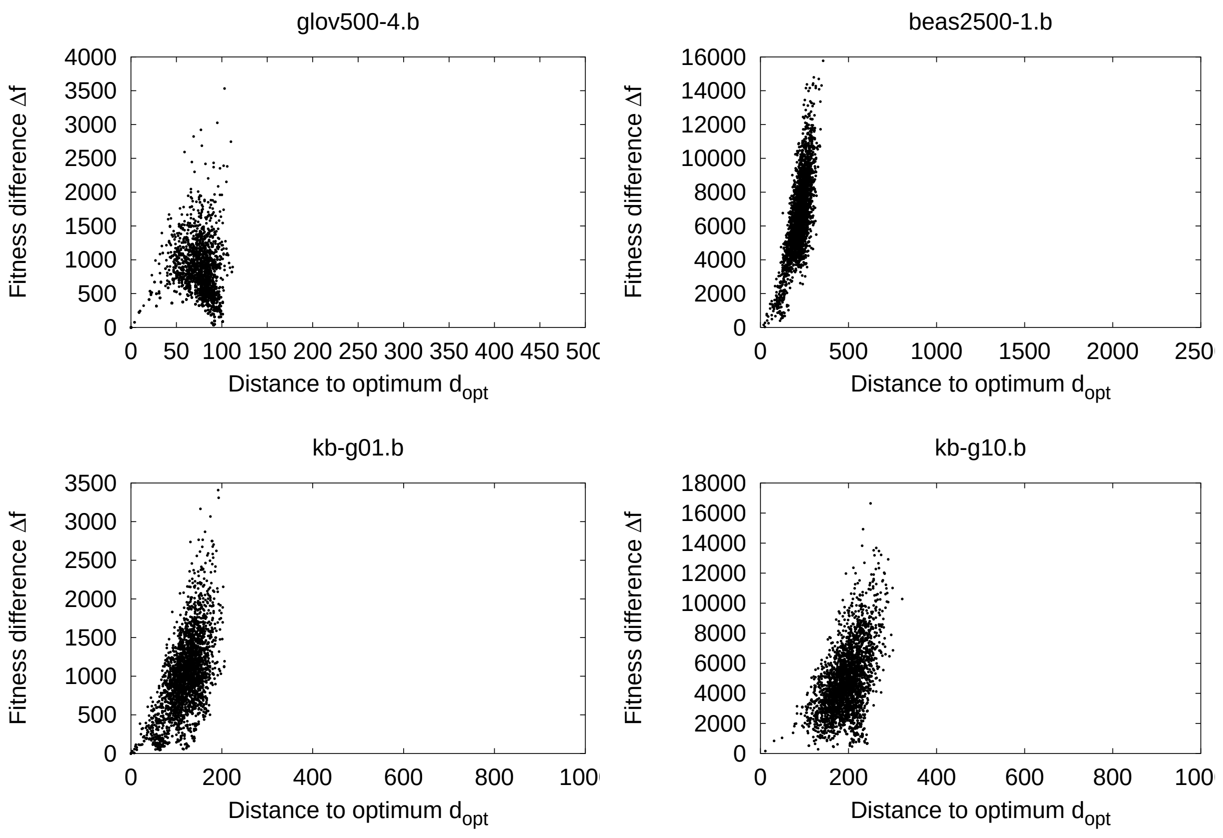


Figure 6.9: Fitness-Distance Plots produced with k-opt local Search

Evaluation of the fitness function: The objective function of the BQP (equation (2.22)) can be used as the fitness function for a genetic algorithm.

Representation: As mentioned before, a solution x to the problem can be represented as a binary string x of length n. All bit combinations are feasible and thus no penalty function is needed.

Crossover: Due to the binary representation, classical genetic operators such as single–point, two–point and uniform crossover can be applied without modification to the BQP.

Mutation: In the GA, a simple bit–flip mutation operator is employed that flips a single bit of the bit string constituting a solution to the BQP.

Selection: There are many different selection techniques for genetic algorithms which can be divided into selection–for–reproduction and selection–for–survival strategies. For example, a random selection scheme for selection–for–reproduction can be used in a GA for the BQP as done in subsequent experiments. Selection–for–survival can be performed by selecting the best out of the pool of parents and newly created offspring, as done in the $(\mu + \lambda)$ evolution strategy [269]. Additionally, it is reasonable to ensure that every solution is contained only once in the population (duplicate checking).

Diversification/restarts: To overcome the problem of premature convergence, the diversification strategy borrowed from the CHC algorithm proposed by Eshelman [88] can be incorporated into the GA: If the algorithm is said to have converged (the average hamming–distance of the population has dropped below a threshold d=10 or there was no change in the population for more than 30 generations), the whole population is mutated except the best individual, and the search is restarted. This kind of selection/diversification strategy has been shown to be very effective for small populations. Essentially, the same diversification mechanism is employed in the MA described below.

The GA for the BQP proposed in this chapter includes the above components and is therefore similar to the CHC algorithm [88] as well as to the MA proposed for the BQP (a local search is not incorporated, of course).

6.4.2 The Memetic Algorithm

A well recognized approach to use domain knowledge in genetic algorithms is the incorporation of local search as found in MAs. Due to the circumstance that all solutions an a population of an MA represent local optima mutation and recombination operators should be modified.

Mutation Operators

The mutation operator is required to flip more than a single bit simultaneously in the solution in order to show effect. The number of bits to change depends on the local search performed subsequently and the properties of the fitness landscape. The hamming distance between parent and mutated offspring should be high enough to minimize the probability that the local search falls back into the same local optimum. On the other hand, it is desired that this jump distance is not too high to prevent the MA from acting like a multi–start local search. In case of a *1-opt* local search, a distance of 3 may be sufficient but for a *k-opt* local search, the jump distance should be considerably higher. At this point, the results of the landscape analysis can help: A distance near the average distance between local optima appears to be a reasonable choice.

Recombination Operators

Recombination should be designed to maximize the distance of offspring and parents. This can be achieved by biasing uniform crossover such as to produce offspring with expected distance of $d/2$ to both parents provided that the hamming distance between the parents is d. Thus, the distance of offspring and parents is maximized by simultaneously obeying respectfulness. This variant of uniform crossover is called HUX [86].

The MAs considered in the experiments include either the mutation operator with a predefined jump distance or the HUX recombination operator. The local search applied to the resulting offspring after recombination is restricted to a region of the search space defined by the two parents: the genes with equal values in the two parents are not modified during local search. Selection and replacement are performed as described above. The simple MAs compared with GAs employ the *1-opt* local search while a sophisticated MA has been evaluated using the randomized *k-opt* local search. Moreover, the greedy heuristic described above is incorporated in the latter MA to produce the starting solutions for the local search in the initialization step of the algorithm.

6.5 Performance Evaluation

Three series of experiments have been performed with the algorithms described above. In a first series, experiments have been conducted to evaluate the performance of the greedy and local search heuristics. A comparison of GAs and MAs with 1-opt local search has been carried out in a second series of experiments. In the last series, the MA with *k-opt* local search has been evaluated on large BQP instances.

6.5.1 Performance of the Greedy and Local Search Heuristics

To evaluate the performance of the greedy and local search heuristics, several experiments were conducted on all 105 problem instances contained in ORLIB [25]. The sets glov-a, glov-b, glov-c and glov-d contain the instances of type a through d described in [116]. The set glov200 (glov500) consists of five problem instances of size $n = 200$ ($n = 500$) and is denoted as type e (f) in [116]. The six sets described in [26] with a density $dens(Q)$ of 0.1 and $n \in \{50, 100, 250, 500, 1000, 2500\}$ consist of 10 instances each. They are denoted beas$\langle n \rangle$. In the experiments, the instances of type g used in [116, 6] with $n = 1000$ and $dens(Q)$ between 0.1 and 1 were also considered. They are denoted as kb-g.

To find a good parameter value for m in the *k-opt* procedure, experiments were performed for beas1000 and beas2500. It appeared that $m = 100$ is a good trade-off between running time and solution quality. For larger values, the running time increased considerably with only small changes in solution quality. Therefore $m = 100$ was chosen in all subsequent *k-opt* local search experiments.

In the experiments, the performance of the randomized greedy algorithm, the *1-opt* local search applied to randomly generated solutions, the fast *k-opt* local search applied to randomly generated solutions, and the combination of the randomized greedy heuristic and fast *k-opt* local search was investigated. To enable a fair comparison, all algorithms implemented in *C++* were run under the same conditions on a Pentium II PC (300 MHz) under the operating system Solaris 2.6.

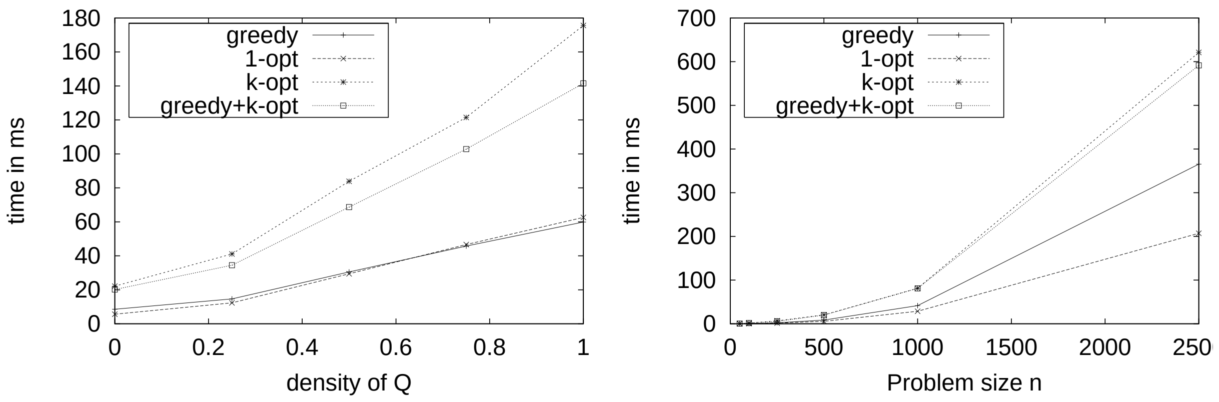


Figure 6.10: Average running times of greedy and local search heuristics for the BQP

In a first experiment, the average running times of the four algorithms and the average solution quality was studied. In Figure 6.10, the average running times (in milliseconds) of 10000 runs of the algorithms are provided. In the left plot, the running times are provided for the five instances of the set glov500 with $n = 500$ and a density $dens(Q)$ contained between 0.1 and 1. As expected, the running time of the algorithms grows linearly with the density of the matrix Q. The running times of the combination of the randomized greedy heuristic and *k-opt* local search are slightly lower than the running times of the *k-opt* local search applied to randomly generated solutions, since the number of iterations of the local search is reduced when it is applied to solutions produced by the greedy heuristic. In the right plot, the running times are provided for the six sets beas50 to beas2500. For all algorithms, the running times grow quadratically with n. The *k-opt* algorithms appear to be 2.5 times slower than the *1-opt* algorithm, and for large n, the greedy heuristic is slower than *1-opt*. Thus, the number of iterations of the *1-opt* procedure grows at most linearly with the problem size for the instances studied.

Table 6.3: Average solution quality of greedy and local search heuristics for the BQP

instances	**greedy**		**1-opt**		**k-opt**		**greedy-k-opt**	
	avg	sdev	avg	sdev	avg	sdev	avg	sdev
glov-a	3.83 %	1.33	2.02 %	0.83	0.38 %	0.30	0.20 %	0.27
glov-b	42.98 %	8.25	29.44 %	5.31	14.69 %	4.54	19.76 %	6.03
glov-c	3.52 %	0.60	1.21 %	0.78	0.24 %	0.24	0.19 %	0.14
glov-d	2.81 %	0.42	2.71 %	0.73	0.71 %	0.35	0.42 %	0.27
glov200	2.41 %	0.77	1.99 %	0.96	0.50 %	0.31	0.31 %	0.15
glov500	1.84 %	0.12	1.95 %	0.36	0.56 %	0.09	0.31 %	0.10
beas50	4.31 %	1.92	5.20 %	3.57	0.89 %	0.82	0.55 %	0.50
beas100	2.37 %	0.94	3.02 %	1.54	0.65 %	0.46	0.49 %	0.56
beas250	2.09 %	0.53	2.44 %	1.12	0.65 %	0.45	0.41 %	0.24
beas500	1.73 %	0.35	2.12 %	0.48	0.62 %	0.23	0.48 %	0.18
beas1000	1.67 %	0.15	1.71 %	0.24	0.54 %	0.12	0.39 %	0.08
beas2500	1.38 %	0.13	1.15 %	0.13	0.40 %	0.07	0.29 %	0.07

The average solution quality of the approaches is displayed in Table 6.3. The solution quality is measured by the average percentage excess (avg) $(100 \cdot (1.0 - f(x)/f_{best}))$ over the best-known solution for a set of up to 10 instances, and the standard deviation (sdev) is also provided for each algorithm. The greedy heuristic shows a better average performance on six of the sets than the *1-opt* local search, while the latter performs better on the remaining six. The *k-opt* heuristic performs considerably better than the greedy and *1-opt* heuristic: with one exception, the average percentage excess is below 1%. However, the combination of greedy and *k-opt* local search performs best on all but one instance with respect to the average quality of the solutions.

In a second experiment, the heuristics were repeatedly applied (multi–start) to show their ability to reach the optimum or best-known solution. Each of the heuristics was started multiple times and the best solution found was returned. For each instance, 30 runs were performed for each algorithm, and the times to reach the best-known solutions were recorded. In Table 6.4, the times for the algorithms that were capable of finding the optimum in all 30 out of 30 runs for all instances in a set are displayed. The average number of repetitions needed (rep), the average time in seconds (avg t) to reach the best-known solution, as well as the maximum time in seconds (max t) to reach the best-known solution is provided. For problems up to a size of $n = 200$, the *1-opt* local search is capable of finding the optimum in

Table 6.4: Time to reach the optimum

instances	**1-opt**			**k-opt**			**greedy-k-opt**		
	rep	avg t	max t	rep	avg t	max t	rep	avg t	max t
glov-a	8.88	0.01	0.04	2.12	0.01	0.03	9.75	0.01	0.14
glov-b	16.89	0.03	0.20	3.11	0.02	0.08	5.67	0.02	0.13
glov-c	9.29	0.01	0.05	1.57	0.01	0.02	2.29	0.01	0.03
glov-d	52.11	0.05	0.33	4.11	0.02	0.08	4.67	0.02	0.11
glov200	217.20	0.40	1.80	12.80	0.08	0.21	10.60	0.07	0.22
beas50	10.70	0.01	0.03	1.70	0.01	0.02	3.00	0.01	0.03
beas100	173.40	0.06	0.81	5.40	0.01	0.05	11.90	0.02	0.15
beas250	–	–	–	20.00	0.09	0.77	10.30	0.05	0.52

less than 2 seconds. For the set glov200 the average number of local searches is about 217. The *k-opt* heuristic needs only about 13 local searches on the average to find the best-known solutions for the instances of this set. Both *k-opt* algorithms perform better on the instances up to $n = 200$; they need less than 0.23 seconds to find the best-known solutions and are able to find the best-known solutions of all the instances of the set beas250. On this set, the greedy *k-opt* combination performs slightly better than the *k-opt* on random solutions: less than 0.53 seconds are needed.

The results show that for small instances (up to $n = 250$), a simple local search is sufficient to find best-known solutions quickly. The more challenging problems have a size of $n = 500$ and higher. The third experiment concentrated on these instances. To enable a fair comparison, the four algorithms were repeatedly applied until a predefined time limit was reached. Once again, 30 runs were performed for each instance. The results are summarized in Tables 6.5. For each instance and algorithm, the average number of repetitions (rep), and the average percentage excess (avg) is given. The time limit in seconds (time) used is provided in the last column of the table.

The greedy heuristic shows to be inferior to the *1-opt* local search: The average percentage excess for the set beas1000 and beas2500 is between 0.727% and 1.170% in case of the greedy heuristic; for the *1-opt* local search the results are between 0.216% and 0.543%. The algorithms based on *k-opt* are considerably better. The worst performance lies 0.128% below the optimum for the beas instances. The kb-g instances appear to be harder; within a time limit of 30 seconds, the average solution quality lies between 0.012 % and 0.489 %. A preference to one of the *k-opt* based algorithms can not be given, since their performance does not differ significantly. Both are able to find the best-known solution for the problems glov500-1 and glov500-2 in all 30 runs, but not for the other problems with $n = 500$ and a density $dens(Q)$ greater 0.25. This indicates that problems with high densities are slightly harder for the *k-opt* local search. However, as the results on the kb-g instances show, the average solution quality is not a simple function of the density: the average percentage access for the problem with density 0.8 (kb-g08) is better than for the problems with density 0.4 and 0.7 (kb-g04 and kb-g07).

The same experiments as for the k-opt local search have been performed for the randomized k-opt local search. The results are presented in Table 6.6. Considering the instances with $n \geq 1000$, the randomized *k-opt* produces solutions with slightly higher objective value than the deterministic *k-opt* local search in the majority of cases (18 out of 30).

Table 6.5: Comparison of greedy, 1-opt, k-opt, and greedy-k-opt on large BQP instances

instances	**greedy**		**1-opt**		**k-opt**		**greedy-k-opt**		**time**
	rep	avg	rep	avg	rep	avg	rep	avg	(sec)
glov500-1	1163	0.472 %	1476	0.100 %	7	0.000 %	7	0.000 %	10
glov500-2	674	0.489 %	798	0.059 %	48	0.000 %	27	0.000 %	10
glov500-3	321	0.796 %	329	0.168 %	85	0.005 %	66	0.001 %	10
glov500-4	211	0.824 %	208	0.170 %	60	0.009 %	41	0.011 %	10
glov500-5	159	0.862 %	153	0.248 %	55	0.003 %	64	0.002 %	10
kb-g01	785	1.039 %	1108	0.581 %	282	0.013 %	276	0.012 %	30
kb-g02	478	1.488 %	574	0.865 %	196	0.046 %	184	0.012 %	30
kb-g03	349	2.739 %	389	1.383 %	88	0.017 %	137	0.085 %	30
kb-g04	275	2.262 %	296	1.148 %	112	0.244 %	131	0.245 %	30
kb-g05	227	1.491 %	235	0.740 %	84	0.022 %	103	0.031 %	30
kb-g06	193	2.810 %	200	1.570 %	72	0.283 %	87	0.188 %	30
kb-g07	168	3.190 %	172	2.012 %	62	0.414 %	76	0.489 %	30
kb-g08	148	2.131 %	150	1.147 %	52	0.183 %	67	0.146 %	30
kb-g09	133	3.083 %	134	1.760 %	48	0.479 %	60	0.320 %	30
kb-g10	120	2.959 %	121	1.788 %	43	0.438 %	54	0.377 %	30
beas1000-1	710	0.766 %	1004	0.221 %	96	0.000 %	246	0.006 %	30
beas1000-2	717	1.086 %	1025	0.351 %	280	0.000 %	340	0.048 %	30
beas1000-3	708	0.890 %	1010	0.239 %	165	0.006 %	289	0.013 %	30
beas1000-4	709	0.813 %	1002	0.216 %	333	0.015 %	337	0.025 %	30
beas1000-5	715	0.984 %	1019	0.348 %	255	0.014 %	360	0.014 %	30
beas1000-6	704	0.727 %	985	0.316 %	236	0.024 %	321	0.037 %	30
beas1000-7	711	0.815 %	1016	0.300 %	367	0.026 %	327	0.033 %	30
beas1000-8	698	1.044 %	991	0.397 %	345	0.051 %	350	0.049 %	30
beas1000-9	712	1.029 %	1011	0.422 %	317	0.014 %	152	0.001 %	30
beas1000-10	719	0.838 %	1020	0.300 %	348	0.008 %	354	0.053 %	30
beas2500-1	164	1.170 %	275	0.541 %	97	0.073 %	100	0.089 %	60
beas2500-2	164	0.864 %	276	0.456 %	97	0.083 %	102	0.106 %	60
beas2500-3	164	1.114 %	274	0.543 %	97	0.053 %	100	0.059 %	60
beas2500-4	164	0.829 %	275	0.390 %	100	0.022 %	93	0.010 %	60
beas2500-5	164	1.051 %	273	0.401 %	98	0.028 %	101	0.036 %	60
beas2500-6	165	0.956 %	275	0.383 %	101	0.091 %	101	0.071 %	60
beas2500-7	164	1.088 %	275	0.527 %	98	0.128 %	99	0.108 %	60
beas2500-8	164	0.749 %	275	0.289 %	102	0.073 %	98	0.024 %	60
beas2500-9	165	0.985 %	273	0.359 %	98	0.063 %	101	0.057 %	60
beas2500-10	165	0.941 %	276	0.450 %	95	0.110 %	101	0.080 %	60

In comparison to the tabu search and simulated annealing for the BQP proposed in [26], the best found solutions obtained with the greedy heuristic and k-*opt* local search for the 10 problems of size 2500 are in 7 out of 10 cases better than the best found solutions reported in [26]. For tabu search and simulated annealing, the running times on a Silicon Graphics Indigo workstation (R4000, 100MHz) are between 12721 and 51873 seconds compared to the 60 seconds for the local search on a Pentium II 300 MHz PC. Thus, the results produced by the k-*opt* local search appear to be superior or at least competitive to the other approaches in

Table 6.6: Randomized k-opt local search on large BQP instances

instance	rep	avg obj	sdev	N_{best}	t (sec)
glov500-1	32	61194.0 (0.000 %)	0.0	30/30	0
glov500-2	44	100161.0 (0.000 %)	0.0	30/30	1
glov500-3	75	138027.2 (0.006 %)	15.5	21/30	5
glov500-4	45	172750.5 (0.012 %)	48.8	22/30	5
glov500-5	54	190502.1 (0.003 %)	2.1	3/30	9
kb-g01	286	131429.0 (0.021 %)	27.8	10/30	23
kb-g02	189	172711.5 (0.044 %)	63.7	2/30	29
kb-g03	80	192533.7 (0.016 %)	66.9	20/30	18
kb-g04	102	215020.7 (0.305 %)	225.7	0/30	30
kb-g05	74	242288.0 (0.033 %)	164.6	9/30	25
kb-g06	65	242522.2 (0.317 %)	310.2	0/30	30
kb-g07	56	252437.8 (0.454 %)	558.3	0/30	30
kb-g08	47	263833.4 (0.164 %)	314.3	4/30	29
kb-g09	43	261595.0 (0.405 %)	543.7	1/30	30
kb-g10	39	273192.9 (0.431 %)	367.4	0/30	30
beas1000-1	182	371434.0 (0.001 %)	9.1	25/30	13
beas1000-2	265	354926.7 (0.002 %)	19.4	19/30	20
beas1000-3	212	371213.7 (0.006 %)	41.1	23/30	16
beas1000-4	295	370628.3 (0.013 %)	50.4	12/30	23
beas1000-5	358	352726.1 (0.010 %)	39.6	2/30	28
beas1000-6	274	359554.4 (0.021 %)	83.1	15/30	21
beas1000-7	344	371096.7 (0.026 %)	69.3	7/30	25
beas1000-8	372	351873.4 (0.034 %)	71.1	1/30	30
beas1000-9	238	349321.0 (0.005 %)	37.3	20/30	18
beas1000-10	297	351397.9 (0.005 %)	44.8	11/30	22
beas2500-1	103	1514987.6 (0.063 %)	456.3	0/30	60
beas2500-2	102	1469920.8 (0.100 %)	384.3	0/30	60
beas2500-3	101	1412953.1 (0.088 %)	440.6	0/30	60
beas2500-4	105	1507080.2 (0.041 %)	408.4	0/30	60
beas2500-5	103	1491470.0 (0.023 %)	190.3	0/30	60
beas2500-6	106	1468076.7 (0.074 %)	401.9	0/30	60
beas2500-7	103	1477232.4 (0.122 %)	566.0	0/30	60
beas2500-8	106	1483541.3 (0.044 %)	242.0	0/30	60
beas2500-9	103	1481734.4 (0.046 %)	365.0	0/30	60
beas2500-10	100	1481874.4 (0.100 %)	470.9	0/30	60

solution quality per time. However, a comparison of the methods under the same conditions (computing hardware, operating system, programming language, coding techniques, ...) is required to support this claim.

6.5.2 Comparison of Genetic Algorithms and Memetic Algorithms

Several experiments were conducted to compare the GAs described above with MAs employing 1-opt local search. The algorithms were applied to 75 different instances contained in OR-

Table 6.7: Comparison of mutation vs. crossover based algorithms for the BQP

				Uniform Crossover			Mutation		
instance	n	dens	best	avg.	N_{opt}	t/s	avg.	N_{opt}	t/s
glov-3a	70	0.11	6037	6036.1 (0.014%)	17/30	38	6036.1 (0.014%)	17/30	30
glov-5a	50	0.21	5737	5734.5 (0.044%)	27/30	12	5735.3 (0.029%)	28/30	6
glov-8b	90	1.00	145	142.9 (1.471%)	22/30	23	145.0 (0.000%)	30/30	1
glov-10b	125	1.00	154	148.1 (3.831%)	20/30	29	154.0 (0.000%)	30/30	5
glov-1d	100	0.11	6333	6293.0 (0.632%)	15/30	33	6324.5 (0.134%)	27/30	8
glov-2d	100	0.22	6579	6543.2 (0.544%)	22/30	26	6559.5 (0.297%)	25/30	14
glov-3d	100	0.30	9261	9217.8 (0.466%)	8/30	44	9238.6 (0.242%)	16/30	28
glov-4d	100	0.41	10727	10713.6 (0.125%)	24/30	23	10718.1 (0.083%)	26/30	8
glov-5d	100	0.50	11626	11599.6 (0.227%)	12/30	44	11608.7 (0.148%)	16/30	29
glov-6d	100	0.61	14207	14163.8 (0.304%)	12/30	37	14172.2 (0.245%)	12/30	36
glov-7d	100	0.70	14476	14433.2 (0.295%)	19/30	28	14466.6 (0.065%)	28/30	9
glov-8d	100	0.80	16352	16317.8 (0.209%)	21/30	21	16352.0 (0.000%)	30/30	5
glov-9d	100	0.89	15656	15610.8 (0.289%)	9/30	42	15590.0 (0.422%)	13/30	34
glov-10d	100	1.00	19102	19064.0 (0.199%)	18/30	32	19102.0 (0.000%)	30/30	2

LIB [25] – 35 small instances ($n \leq 125$) [116], 10 instances of medium size ($n = 200, n = 500$) [116], and 30 large instances with n up to 2500 [26]. The population size was set to $P = 100$ in case of the genetic algorithms without local search, and in the presence of local search a population size of $P = 40$ was used. For all experiments, the recombination/mutation (application) rate was set to 0.5, i.e., $0.5 \cdot P$ offsprings were created in each generation. During the restarts, the individuals were mutated by flipping a third of all the bits in the bit vector.

The genetic algorithms described above, including the local search heuristic, were implemented in C++. All experiments were performed on a Pentium II PC (300 MHz) under the operating system Solaris 2.6. The algorithms were terminated when the optimum/best-known solution was found or a predefined time limit was reached.

In a first experiment, the genetic algorithm with uniform crossover as well as a genetic algorithm solely based on mutation were run on the set of 35 small instances. The time limit was set to 60 seconds per run. The results are shown in Table 6.7. In the first four columns, the name of the instance, the problem size n, the density (dens) of matrix Q, and the fitness of the optimum or best-known solution (best), respectively, is provided. The fifth and the sixth columns contain the results for two genetic algorithms, the first using uniform crossover and the second using the bit-flip mutation operator with a mutation rate of $1/n$ per gene. These columns contain the average final fitness values (avg.) out of 30 runs, the number of runs in which the optimum/best-known solution could be found (N_{opt}), and the average runtime of the algorithm in seconds (t/s). Only those instances are listed for which the best-known solution could not be found in *all* the runs. For the remaining 21 instances, the optimum/best-known solution could be found in 1 – 8 seconds.

The results in Table 6.7 show that mutation based search is superior to crossover based search for all instances. The application of uniform crossover yields better average fitness than mutation in case of glov-9d, but the mutation GA finds the best-known solution more often. The amount of epistasis in the problem expressed by the density of the matrix Q does not seem to influence the "hardness" of an instance since the algorithms do not perform significantly better on instances with low epistasis: other characteristics of the problems seem to have a higher influence. The hardest problems are the problems of type d (glov-1d

Table 6.8: Comparison of genetic algorithms and memetic algorithms

			Uniform Crossover			**Mutation**			**MA**		
instance	dens	best	avg. p	N_{opt}	t/s	avg. p	N_{opt}	t/s	avg. p	N_{opt}	t/s
glov200-1	0.1	16464	0.154	15/30	38	0.117	15/30	32	0.000	30/30	<1
glov200-2	0.2	23395	0.156	2/30	56	0.106	14/30	36	0.000	30/30	<1
glov200-3	0.3	25243	1.036	11/30	45	0.980	11/30	41	0.000	30/30	<1
glov200-4	0.4	35594	0.240	5/30	51	0.075	12/30	37	0.000	30/30	<1
glov200-5	0.5	35154	0.610	3/30	55	0.526	10/30	42	0.000	30/30	<1
glov500-1	0.1	61194	0.689	0/30	120	0.359	2/30	115	0.000	30/30	3
glov500-2	0.25	100161	0.802	0/30	120	0.361	0/30	120	0.000	30/30	5
glov500-3	0.5	138035	1.369	0/30	120	0.509	1/30	116	0.000	30/30	32
glov500-4	0.75	172771	1.291	0/30	120	0.446	2/30	117	0.077	0/30	120
glov500-5	1.0	190507	2.500	0/30	120	0.775	0/30	120	0.002	9/30	103

– glov-10d), and the corresponding densities have a wide range (from 0.11 up to 1.0).

To investigate the scalability of the algorithms, a second experiment was performed for 20 medium sized instances with varying densities. The time limit was set to 60 seconds for the problems of size $n = 200$, and to 120 seconds for the problems of size $n = 500$. The results for the GA with uniform crossover, the GA with mutation, and the MA are shown in Table 6.8.

Again, the density of matrix Q and the best-known solutions are provided for each instance in the second and third columns. For each algorithm, the average deviation from the optimum (avg. p) is given in percent ($p(x) = 100 \cdot (1.0 - f(x)/f_{best})$), the number of times ($N_{opt}$) the optimum/best-known solution was found, and the average running time in seconds (t/s) is provided. Again, mutation-based search outperforms crossover-based search, but the memetic algorithm (MA) proves to be much more effective. For the instances of size $n = 200$, the running times to reach the best-known solution are below one second on the average. Problem glov500-4 appears to be hard for the MA: in the time limit of 120 seconds, the best-known solution could not be found. However, allowing longer running times (2000 seconds), our algorithm was able to find the best-known solution, too (in 3 out of 30 times). Problem glov500-5 also seems to be a harder problem, although in comparison to glov500-4 the best-known solution could be found in 9 out of 30 times.

In order to put the results into perspective, the MA with *1-opt* local search was applied to larger problems of size 500, 1000, and 2500. For these problems, the performance of two alternative approaches, tabu search and simulated annealing, has been reported in [26]. Table 6.9 displays the results of the memetic algorithm on 30 large problems (all with a density of 0.1) in comparison to the results reported in [26].

For the MA, the average number of generations (gen), the best solution value found (best) and the average final solution over 30 runs (avg.), as well as the runtime in seconds (t/s) are shown. For the tabu search (TS) and simulated annealing (SA) algorithms, the best solution found and the total running times on a Silicon Graphics Indigo (R4000 CPU with 100 MHz) are provided. The dots indicate the approach for eacvh instance that found the best-known solution. The figures demonstrate the effectiveness of the proposed MA approach: for 14 out of the 30 problems, new best solutions could be found, and only for 4 problems one of the competitors found a slightly better solution. However, the comparison should be treated with care since the running times are not directly comparable due to the

Table 6.9: Comparison of three algorithms for the BQP

	MA				**TS**		**SA**	
instance	gen	best	avg.	t/s	best	t/s	best	t/s
beas500-1	4361	• 116586	116586.0	120	• 116586	956	• 116586	1006
beas500-2	4611	• 128339	128339.0	120	128223	979	128204	1009
beas500-3	4336	• 130812	130812.0	120	• 130812	987	• 130812	1030
beas500-4	4723	• 130097	130095.7	120	• 130097	1003	130077	1061
beas500-5	4511	• 125487	125487.0	120	• 125487	964	125315	1030
beas500-6	4565	• 121772	121770.2	120	121719	966	121719	1028
beas500-7	4378	• 122201	122201.0	120	• 122201	952	• 122201	1014
beas500-8	4725	• 123559	123535.8	120	• 123559	1006	123469	1050
beas500-9	4668	• 120798	120789.8	120	120797	954	• 120798	998
beas500-10	4422	• 130619	130619.0	120	• 130619	971	• 130619	1012
beas1000-1	3824	• 371438	371304.1	600	• 371438	4608	371134	7150
beas1000-2	3834	• 354932	354862.3	600	• 354932	4514	354637	6794
beas1000-3	3829	• 371236	371233.8	600	371073	4518	371226	6943
beas1000-4	3819	• 370675	370506.0	600	370560	4580	370265	7011
beas1000-5	3796	352730	352687.6	600	• 352736	4512	352297	6939
beas1000-6	3687	• 359629	359487.8	600	359452	4444	359313	6749
beas1000-7	3927	• 371193	371084.9	600	370999	4546	370815	6885
beas1000-8	3491	• 351994	351844.6	600	351836	4461	351001	6961
beas1000-9	3910	• 349254	349253.3	600	348732	4488	348309	6626
beas1000-10	4100	351408	351125.6	600	351408	4474	• 351415	6734
beas2500-1	998	• 1515306	1514804.6	1200	1514971	52011	1515011	63080
beas2500-2	992	• 1470470	1469721.0	1200	1468694	51659	1468850	62787
beas2500-3	1019	• 1413671	1412943.0	1200	1410721	49101	1413083	60963
beas2500-4	978	• 1507701	1507674.2	1200	1506242	50642	1506943	63018
beas2500-5	982	• 1491796	1491623.4	1200	• 1491796	51194	1491465	63470
beas2500-6	987	• 1468427	1467918.2	1200	1467700	51669	• 1468427	63310
beas2500-7	993	1478297	1477101.7	1200	1476059	50798	• 1478654	62833
beas2500-8	1012	1483702	1483226.9	1200	• 1484199	49861	1482953	61918
beas2500-9	989	• 1482413	1481622.9	1200	1482306	51873	1481834	62978
beas2500-10	1010	• 1482733	1481899.2	1200	1482354	50981	1482166	62777

different hardware/software platforms used.

Another evolutionary approach by Lodi et al. is reported in [194]. The approach is similar to the *1-opt* local search MA in that all individuals in the population represent local optima. Recombination is performed by utilizing the MinRange algorithm [244] instead of our simple form of crossover. It is unclear how much the heuristic crossover of the authors contributes to the performance of their algorithm. A direct comparison is not possible since they used a different platform and do not provide the average final solution values. For example, the (shortest) running time to reach the best-known solution of instance glov500-1 is reported to be 43.06 seconds on a Silicon Graphics INDY R10000sc with 195 MHz. The MA took 3 seconds *on the average* on a Pentium II PC with 300 MHz to find the same solution value.

Table 6.10: Comparison of mutation and recombination-based MAs for the BQP

	MA-MUT				MA-HUX			
instance	gen	avg. (quality)	N_{opt}	t/s	gen	avg. (quality)	N_{opt}	t/s
glov500-1	0	61194.0 - 0.000%	20/20	1	0	61194.0 - 0.000%	20/20	1
glov500-2	0	100161.0 - 0.000%	20/20	2	0	100161.0 - 0.000%	20/20	2
glov500-3	1	138035.0 - 0.000%	20/20	3	9	138035.0 - 0.000%	20/20	4
glov500-4	0	172771.0 - 0.000%	20/20	5	57	172764.9 - 0.004%	18/20	27
glov500-5	7	190507.0 - 0.000%	20/20	23	141	190505.0 - 0.001%	12/20	68
kb-g01	9	131456.0 - 0.000%	20/20	13	252	131455.1 - 0.001%	19/20	55
kb-g02	7	172788.0 - 0.000%	20/20	18	375	172785.5 - 0.001%	16/20	146
kb-g03	3	192565.0 - 0.000%	20/20	15	192	192563.5 - 0.001%	19/20	102
kb-g04	46	215441.0 - 0.110%	11/20	162	438	215127.6 - 0.256%	0/20	300
kb-g05	2	242367.0 - 0.000%	20/20	20	260	242358.5 - 0.003%	8/20	210
kb-g06	14	243293.0 - 0.000%	20/20	84	324	243017.6 - 0.113%	0/20	300
kb-g07	24	253446.9 - 0.056%	13/20	151	255	253113.0 - 0.188%	1/20	296
kb-g08	15	264216.2 - 0.020%	15/20	112	206	264042.9 - 0.085%	4/20	257
kb-g09	18	262597.4 - 0.023%	16/20	155	196	262093.0 - 0.215%	0/20	300
kb-g10	29	274289.7 - 0.031%	5/20	253	186	273814.5 - 0.204%	0/20	300
beas2500-1	11	1515944.0 - 0.000%	20/20	101	428	1515761.9 - 0.012%	0/20	600
beas2500-2	65	1471243.4 - 0.010%	10/20	488	365	1470487.4 - 0.061%	1/20	583
beas2500-3	13	1414192.0 - 0.000%	20/20	114	422	1413645.9 - 0.039%	0/20	600
beas2500-4	4	1507701.0 - 0.000%	20/20	52	154	1507671.4 - 0.002%	14/20	234
beas2500-5	15	1491816.0 - 0.000%	20/20	130	433	1491688.8 - 0.009%	0/20	600
beas2500-6	24	1469162.0 - 0.000%	20/20	191	423	1468727.8 - 0.030%	0/20	600
beas2500-7	37	1479004.4 - 0.002%	19/20	289	401	1478354.9 - 0.046%	0/20	600
beas2500-8	7	1484199.0 - 0.000%	20/20	71	407	1483916.6 - 0.019%	1/20	572
beas2500-9	43	1482408.8 - 0.000%	14/20	324	403	1482173.5 - 0.016%	2/20	570
beas2500-10	36	1483286.2 - 0.005%	16/20	281	410	1482570.8 - 0.053%	0/20	600

6.5.3 A Memetic Algorithm with *k-opt* Local Search

In a final series of experiments, the MA employing randomized *k-opt* local search was investigated. In the experiments, the population size was set to 40, and the diversification rate to $n/3$. The variation operator application rate was set to 0.5, i.e., 20 offspring were generated per generation. Two variants of the MA, one using mutation and one using recombination to achieve variation in the evolutionary search were considered. In the mutation-based MA (MA-MUT), the mutation operator produces offspring with a distance of $d = n/10$ to the parent (a 10th of the bits in the genome are flipped). The recombination-based MA relies on the HUX recombination operator (MA-HUX). To evaluate the performance of the algorithms, runs were performed on the five instances of set glov500, the ten instances of set kb-g used in the search space analysis above, and on the 10 instances of the set beas2500. A time limit was chosen for each set: the algorithms were terminated as soon as the best-known solution was found or after 120 seconds in case of the first set, after 300 seconds in case of set kb-g, and after 600 seconds in case of the third set. All running times are provided for a Pentium II PC with 300 MHz. The results are summarized in Table 6.10: For each instance and

Table 6.11: Best-known Solutions of large BQP Instances

instance	MA *k-opt*	MA *1-opt*	TS/SA
beas1000-1	371438	• 371438	• 371438
beas1000-2	354932	• 354932	• 354932
beas1000-3	371236	• 371236	371226
beas1000-4	370675	• 370675	370560
beas1000-5	• 352760	352730	352736
beas1000-6	359629	• 359629	359452
beas1000-7	371193	• 371193	370999
beas1000-8	351994	• 351994	351836
beas1000-9	• 349337	349254	348732
beas1000-10	351415	351408	• 351415
beas2500-1	• 1515944	1515306	1515011
beas2500-2	• 1471392	1470470	1468850
beas2500-3	• 1414192	1413671	1413083
beas2500-4	1507701	• 1507701	1506943
beas2500-5	• 1491816	1491796	1491796
beas2500-6	• 1469162	1468427	1468427
beas2500-7	• 1479040	1478297	1478654
beas2500-8	1484199	1483702	• 1484199
beas2500-9	1482413	• 1482413	1482306
beas2500-10	• 1483355	1482733	1482354

algorithm, the average number of generations evolved (gen), the average final solution value (avg) together with the percentage access (quality), the number of times the best-known solution was found (N_{opt}), and the average running time in seconds (t/s) is provided.

Although the landscape of the BQP instances considered are highly correlated, the recombination-based MA performs significantly worse than the MA with mutation. The local optima appear to be too close to each other for a recombination operator to be effective. In an additional experiment, this claim could be supported: After the majority of recombinations, the local search produces a solution equal to one of the parents. Thus, recombination-based search is ineffective for this type of landscape despite of the high correlation of fitness and distance to the optimum.

Compared to the MA based on *1-opt* local search, new best-known solutions could be found for 7 of the 10 instances of the set beas2500 in shorter time. In Table 6.11, the (new) best-known solution values for large instances are shown. The dot in each line indicates which algorithm discovered the best-known solution.

The MA employing mutation appears to be highly effective since even for large instances, the best-known solution can be found in 52 up to 488 seconds (on average). for the set kb-g, the best-known solution reported in [6] could be found for each instance in short time ranging from 13 seconds on average for the instance with density 0.1 to five out of 20 times within 300 seconds for the instance with density 1.0. The set glov500 is easily solved to optimality within less than 24 seconds on average by MA-MUT.

In comparison, the tabu search utilizing critical event memory [115] found the best-known solution only for 7 out of the 10 instances of set kb-g, in a CPU time of four minutes on a Pentium 200 PC. The scatter search approach proposed in [6] found the best-known solutions of all problems in this set, but no CPU times were reported. Recently, a highly effective simulated annealing approach has been devised for the BQP [166]. In very short time, the best-known solutions could be found for the sets beas1000 and beas2500, but the average solution quality is worse in comparison to the MA. The authors therefore proposed a MA similar to the one presented here showing a similar performance [167]. Hence, memetic algorithms prove to be very robust and effective search algorithms for the BQP.

6.6 Summary

In this chapter, a random-walk correlation analysis as well as a fitness distance correlation analysis has been employed to gain insight into the structure of the BQP. Although BQP instances have resemblance to *NK*-landscapes with non-uniform fitness distribution functions, they exhibit totally different properties. In *NK*-landscapes, with increasing epistasis the landscape becomes more and more rugged. This phenomenon could not be observed in the studied BQP instances. Instead, landscape ruggedness expressed by the correlation length of the landscapes tends to decrease slightly with increasing epistasis. Furthermore, the local optima with respect to *1-opt* and a newly proposed *k-opt* local search are shown to lie in a small fraction of the search space. The local optima produced by the latter local search have an average distance of a 5th to the 10th of the diameter of the landscape. The correlation of the objective value of the local optima and their distance to the global optimum or best-known solution is also shown to be very high.

A memetic algorithm for the BQP has been proposed, and in experiments it was compared to genetic algorithms without local search. The results clearly demonstrate the superiority of the memetic approach even if the simple *1-opt* local search is incorporated: beginning at a problem size of $n = 200$, the MA outperforms the GA by far. However, it is shown that for such small instances the *k-opt* local search is capable of finding the optimum in less than a second. For larger instances, a MA is devised incorporating randomized *k-opt* local search. Although the landscapes of the studied BQP instances are highly structured with respect to the distribution of the local optima, it turned out that a recombination-based MA is not able to exploit this property since the local optima are too close together hindering the recombination in exploring new local optima. Hence, a MA is proposed for large instances employing a mutation operator with a mutation jump distance derived from the landscape analysis: The jump distance is set to a tenth of the problem size and is thus close to the average distance of the local optima. The performance of the mutation-based MA has been evaluated on several large instances, and it was shown that better solutions compared to the MA using *1-opt* local search have been found with a high frequency in shorter time. Thus, the MA proves to be among the best performing heuristics for the BQP especially when high average solution quality for large instances is desired.

However, due to the fact that the local optima of the considered instances are close together in terms of the hamming distance, it may be more promising to focus on neighborhood search techniques such as simulated annealing and tabu search in combination with k-opt neighborhoods which can be designed to overcome the small barriers between the local optima by considering appropriate down-hill moves. Other instances, which are not purely random in nature or represent transformed problems, may not have the property that the local optima are located in such a small region of the search space. Here, MAs may be required to achieve high quality solutions.

Chapter 7

The Traveling Salesman Problem

7.1 Introduction

The traveling salesman problem (TSP) is one of the best-known combinatorial optimization problems. It can be stated as follows: Given n cities and the geographical distance between all pairs of these cities, the task is to find the shortest closed tour in which each city is visited exactly once. In the last years, the exact solution of large TSP instances has made an enormous progress due to the improvement of branch & cut algorithms. Furthermore, the TSP has been widely used as a problem for testing new heuristic algorithms and general purpose optimization techniques. As a result, highly effective heuristics have been proposed that a capable of solving TSPs with thousands of cities.

In this chapter, memetic algorithms for the TSP are introduced which have been shown to belong to the best heuristics currently available for the TSP. A landscape analysis is performed to identify properties of TSP instances which can be exploited by MAs. It is shown that although all TSP instances share certain characteristics, there are some landscapes that differ significantly from others, leading to a different performance of heuristic approaches. However, the analysis reveals that respectful recombination is capable of exploiting the distribution of local minima in the TSP landscape.

A new generic greedy recombination operator is described that is used to identify important properties of recombination operators in memetic algorithms for the TSP. Various recombination operators are compared in experiments, and it is shown that many operators show similar performance. On the other hand, it is shown – as expected due to the results of the landscape analysis – that recombination needs to be respectful to be highly effective.

Furthermore, it is demonstrated that the successor of the already published MA [106, 210] is capable of finding optimum solutions for problems up to 3795 cities in short time and is thus superior to any other evolutionary algorithm for the TSP known to the author. In additional experiments, it is shown that with the approach problems of up to 85900 cities can be tackled.

The memetic algorithms based on DPX have been published in [105, 106, 210, 214]. However, the results reported in this chapter have been improved considerably.

7.2 Heuristics for the TSP

Because of its simplicity and applicability, the TSP has for decades served as an initial proving ground for new ideas to solve combinatorial optimization problems. Besides the fast development in solving TSP instances to optimality, an enormous progress has been made in the field of heuristics.

Most of the earliest algorithms belong to the class of construction heuristics. Examples of this class are *nearest neighbor heuristics* and *insertion heuristics*, for which a detailed description and comparison can be found in [219, 261]. Another intuitive approach is the *greedy heuristic*, also known as the *multi-fragment heuristic* [158, 27]. Furthermore, there are more elaborate tour construction heuristics, such as the *Christofides algorithm* [53] which is based on spanning trees, or the *savings heuristic* also known as *Clarke and Wright algorithm* originally developed for the vehicle routing problem [54]. However, these heuristics perform poor in comparison to *local search heuristics* which belong to the class of improvement heuristics. But, instead of applying a local search to randomly generated solutions, a local search can be applied to solutions generated by a (randomized) tour construction heuristic. Surprisingly, not the best performing construction heuristic is best suited in combination with local search, as shown by several researchers independently [29, 158, 219, 261]. For example, in [158], it is shown that although the savings heuristics performs better than the greedy and nearest neighbor heuristics, in combination with *2-opt* or *3-opt* local search it performs worst (even worse than the local search applied to random solutions). In fact, the best suited construction heuristic is the greedy heuristic, as shown in [29, 158]. It appears that the starting tour for a local optimization must contain a number of exploitable defects, i.e., some rather long edges, and if a tour is too good it may not have these.

Since greedy and local search heuristics are among the most efficient algorithms for the TSP with short running times and thus the most interesting for incorporation into an MA, these two types of algorithms are described in the following paragraphs in more detail. Many other heuristics have been proposed for the TSP including simulated annealing [303, 153], tabu search [90], ant colonies [112, 78, 287], artificial neural networks [253, 223, 81], search space smoothing [130], perturbation [55], and evolutionary algorithms [232, 311, 103, 95, 143, 299, 305, 302].

7.2.1 The Greedy Heuristic

Although the nearest neighbor heuristic can be regarded as a greedy heuristic, the term is usually used for the following variant of the greedy algorithm.

This heuristic can be viewed as considering the edges of the graph in increasing order of length, and adding any edge that will not make it impossible to complete a tour. Thus, the algorithm builds up a TSP tour for N cities (a cycle of length N in a graph) by adding one edge at a time, starting with the shortest edge, and repeatedly adding the shortest remaining available edge. In the algorithm, an edge is referred to as available if it is not yet in the tour and if adding it would not create a degree-3 vertex or a cycle of length less than N.

While in the nearest neighbor heuristic, partial tours maintain a single TSP fragment, the greedy heuristic employs a set of fragments. Therefore, the greedy heuristic is also known under the name *multi-fragment heuristic* [29].

The implementation sketched above requires $O(N^2 \log N)$ time. However, using appropriate data structures, the running time of the algorithm can be reduced considerably. As shown in [29], using K-d trees to calculate nearest neighbors [28], and using a priority queue to store available candidate edges, the running time is reduced to $O(N \log N)$ for uniform data (Euclidean TSP with points uniformly distributed in the unit square).

7.2.2 Local Search

Local search algorithms for the TSP are based on simple tour modifications. A local search algorithm is specified in terms of a class of operations called moves that can be used to transform one tour to another. We can view local search as a neighborhood search process where each tour has an associated neighborhood of tours, i.e., those that can be reached by a single move. The search algorithm repeatedly moves to a better neighbor until no better neighbors exist. Moves proposed for the TSP can be divided into *node exchange operators*, *node insertion operators* and *edge exchange operators*.

Viewing a TSP tour as a sequence of cities which defines the order in which to visit the cities, the *node exchange* operator simply exchanges two nodes in the sequence.

The *node re-insertion* operators work by deleting a node from a tour and inserting it at another place in the tour. Variations of this scheme exist in which two nodes are reinserted (edge insertion) [261] or up to 3 nodes are reinserted (Or-opt) [241].

Among simple local search algorithms, the most famous are *2-opt* and *3-opt* [191] which are examples of edge exchange algorithms. The 2-opt algorithm was first proposed by Croes [59], although the basic move had already been suggested by Flood [94]. This move deletes two edges thus breaking the tour into two paths, then reconnects those paths in the other possible way as shown in Figure 7.1. In the example, the edges $(0, 2)$ and $(5, 6)$ are exchanged

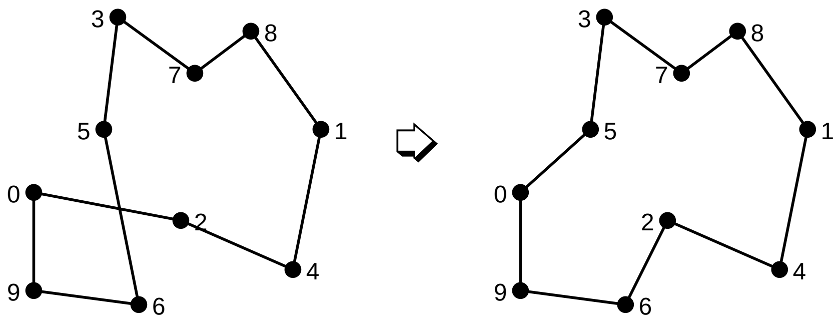


Figure 7.1: Neighborhood search by exchange of two edges

by the edges $(0, 5)$ and $(2, 6)$.

Analogously, in 3-opt, up to three edges exchanged. With 3-opt the neighborhood size increases from $O(N^2)$ to $O(N^3)$ compared to the other local searches. In practice, however, the running time for searching the neighborhood can be reduced so that 3-opt is applicable even to large instances. For example, nearest neighbor lists are used to restrict the number of candidate edges for the replacement of edges in an exchange [261, 219, 158]. Furthermore, the concept of *don't look bits* proposed by Bentley [27] reduces the search time for an improving move considerably. Compared to other neighborhood search algorithms such as tabu search, the first improving move is accepted in this scheme rather than the best.

It has been shown that edge exchange local search is much more effective than node re-insertion or node exchange [261, 219]. Generally, the higher the k of a k-opt local search, the better the resulting tours. However, since the neighborhood size grows exponentially with k only small k turn out to be practical. Lin and Kernighan have shown that a subset of the k-opt neighborhood can be searched very efficiently with considerable decrease in tour length compared to 2 or 3-opt.

The Lin-Kernighan Algorithm

For over a decade and a half, from 1973 to about 1989, the world champion heuristic for the TSP was generally recognized to be the local search algorithm of Lin and Kernighan (LK) [192]. This algorithm is both a generalization of 3-opt and an outgrowth of ideas the same authors had previously applied to the graph partitioning problem [171].

The basic idea of the LK algorithm is to build up complex moves by combining simple sub-moves to exchange a variable number of edges. The sub-moves usually employed are 2-opt and 3-opt moves although variants exist where node re-insertion and 2-opt has been used [261]. To illustrate the behavior of the heuristic, an example of an edge exchange is shown in Figure 7.2. (In the figure, a TSP tour is displayed as a circle and the length of

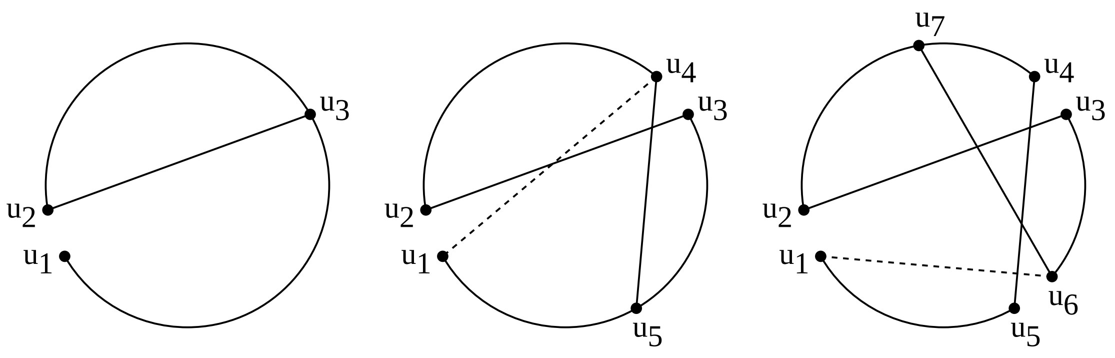


Figure 7.2: An edge exchange in the LK heuristic

the edges do not resemble their length in the TSP graph.) Briefly, the LK heuristic can be described as follows. In each step, we have a situation where the tour is broken up at one node forming a 1-tree (a spanning tree with an extra edge) as shown on the left of the figure. This 1-tree can be easily transformed into a feasible TSP tour by breaking up one edge of the degree-3 vertex and connecting the two degree-1 vertices. Consider now the example in which an improving k-exchange is searched beginning with node u_1. First, the edge (u_1, u_2) is replaced by a shorter edge (u_2, u_3). Now, the algorithm considers to close up to a tour by connecting the predecessor of u_3 called u_4 with u_1 and thus replacing edge (u_3, u_4) with edge (u_4, u_1). In this case, we made a 2-change since we replaced the edges (u_1, u_2) and (u_4, u_3) with (u_2, u_3) and (u_4, u_1). Alternatively, we can replace the edge (u_3, u_4) with (u_4, u_5) resulting in a new 1-tree. Once again, we may close up to a tour by connecting u_6 with u_1 or continue searching by connecting u_6 to another node u_7 as shown in the right of the figure. Thus, the heuristic performs sequential changes of edges until no further exchanges are possible or favorable to find the best k-change in an iteration. The number of exchanges that are tried is bound by the gain criterion which is fulfilled if the gain of replacing k edges with new edges without closing up to a tour is above zero. The change made in an iteration is the one with the highest gain when closing up to a tour. If the search for an improving k-change fails, several levels of backtracking are considered. For example, alternatives for (u_2, u_3) at the first level and alternatives for (u_4, u_5) at the second level are considered.

A more detailed description of the LK algorithm would go beyond the scope of this thesis and can be found in the original paper by Lin and Kernighan [192].

A major drawback of the LK heuristic besides the high effort needed for its implementation is its rather long running time. Therefore, several improvements to the original algorithm have been made such as candidate lists based on nearest neighbors, and don't look bits [158]. Furthermore, efficient data structures have been proposed to perform the sub-moves since they consume most of the running time of the algorithm especially for large TSP instances ($N > 1000$) [201, 104].

7.2.3 Evolutionary Algorithms

Various attempts have been made to apply evolutionary algorithms to the TSP. For example, evolutionary programming has been applied to the TSP by Fogel using node re-insertion as mutation operator [96] and random 2-opt moves (random exchanges of two edges) [95]. Evolution strategies have been applied to the TSP by Herdy [136] and Rudolph [266]. While Herdy conducted experiments with node exchange, node re-insertion, and edge exchange operator (two and three edges), Rudolph chose a real vector representation for the TSP and applied the ES on continuous variables. The majority of publications, however, deals with representations and/or recombination operators for GAs for the TSP.

Besides the most commonly used *path representation* [118, 69, 239, 121] in which a tour is coded as a vector of discrete variables of length N that provides the order in which to visit the cities and is thus a permutation π of the set $\{1, \ldots N\}$, other representations have been proposed such as the *adjacency representation* [124], the *adjacency matrix representation* [143], the *precedence matrix representation* [103], *the ordinal representation* [124], and the *edge list representation* in combination with the path representation [311].

There is enormous number of recombination operators for the TSP, most of which have the disadvantage that they do not scale well or they are only effective in combination with additional heuristic operators. The reason will be illustrated by an example.

The partially–mapped crossover (PMX) has been introduced by Goldberg and Lingle [118]. It performs a sequence of swapping operations to create offspring. Firstly, a mapping section is chosen at random. In the example below, the mapping section is marked by '|':

Partially–Mapped Crossover												
Parent A	0	9	6	|	5	3	7	8	|	1	4	2
Parent B	0	5	3	|	7	4	1	8	|	2	6	9
Offspring A'	0	9	6	|	7	4	1	8	|	5	3	2
Offspring B'	0	1	4	|	5	3	7	8	|	2	6	9

Secondly, in parent A, cities 5 and 7 are swapped, then 3 and 4, and at last cities 5 and 1. Now, the mapping section is equal to the mapping section of parent B. Third, parent B has to be transformed analogously by a sequence of swaps, these are: 5 and 7, 3 and 4, 6 and 5, and 1 and 7. The resulting offspring are shown above. Both offspring A' and B' contain three and two edges not shared by the parents, respectively. For example, offspring A' is a feasible solution but does not consist entirely of genetic material from its parents: The edges $(6, 7)$, $(5, 8)$, and $(2, 3)$ are not contained in either of the parents. Figure 7.3 displays the tours of parent A, parent B, and offspring A'. The tour lengths are 1707.96, 1834.27, and 2251.00, for parent A, parent B and child A', respectively. Here, the crossover has disruptive effects: although only three edges are included that are not contained in the parents, the tour length is considerably longer than the lengths of the parent tours. Note that the edges of the parents in the example above can be recombined to the solution of length 1507.94.

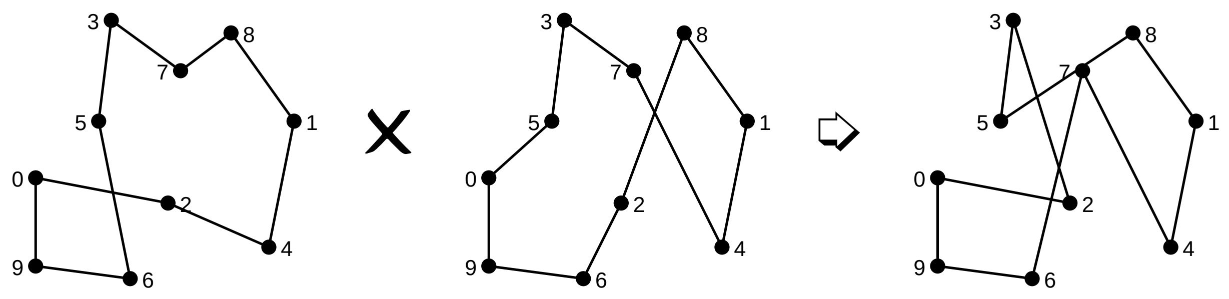


Figure 7.3: Crossover of TSP tours (PMX)

The introduction of foreign edges into the child is referred to as *implicit mutation* and has a high impact on the effectiveness of recombination operators. If the number of foreign edges gets to high, the GA degrades to pure random search. But even a small number of foreign edges can prevent a GA from finding (near) optimum solutions, since these edges can be arbitrarily long and thus may have a high impact on the objective value. In other words, the objective values of parents and offspring may not have a high correlation.

The phenomenon of implicit mutation during recombination can be observed by almost all recombination operators for the TSP. Grefenstette [125] concludes from his studies:

> *"Finally, it's widely recognized that GAs are not well suited to performing finely tuned local search. [...] Once the high performance regions of the search space are identified by the GA, it may be useful to invoke a local search routine to optimize the members of the final population."*

As a consequence, many researchers incorporated greedy choices into their recombination operators and/or used a local improvement techniques to achieve better results [125, 291, 190, 154]. Beginning with Brady [39], many researchers have made consequent use of local search in their evolutionary algorithms for the TSP. In the following, these approaches – which can be classified as memetic algorithms – are briefly described. They have been shown to be among the best heuristic techniques for the TSP.

The Evolutionary Algorithm of Brady

One of the earliest evolutionary approaches for the TSP using local search is the evolutionary algorithm of Brady [39]. In his approach, the solutions produced by crossover are optimized with a local search he calls *quenching* since it can be regarded as a *zero temperature simulated annealing*. In this evolutionary algorithm, TSP tours are coded using the path representation. A sub-path in one parent is sought that has a corresponding sub-path in the other parent containing the same cities. If the sub-path is shorter than the corresponding sub-path, this sub-path in the other parent is replaced:

Brady's Crossover												
Parent A	4	2	0	|	8	9	5	3	7	|	6	1
Parent B	2	|	9	8	3	7	5	|	0	1	6	4
Offspring A'	4	2	0	|	9	8	3	7	5	|	6	1

In the above example, the sum $d_{89}+d_{95}+d_{53}+d_{37}$ is greater than sum $d_{98}+d_{83}+d_{37}+d_{75}$, hence the sub-path 9,8,3,7,5 form B is copied over to A (see A') overwriting path 8,9,5,3,7. The path in parent B remains unchanged. Brady reports that for a 64-city problem it was best to search for sub-paths of length between 8 and 32. A disadvantage of this approach is that it is quite expensive to search for possible crossing-over points.

With this scheme only up to two foreign edges are copied to the parents. In the above example, the edges are (0,9) and (5,6).

Brady's algorithm can be regarded as the first memetic algorithm proposed for the TSP.

ASPARAGOS

The *Asynchronous Parallel Genetic Optimization Strategy (ASPARAGOS)* [121, 122] has been the best evolutionary strategy for the TSP for years. In this approach, offspring are generated using *Maximum Preservative Crossover (MPX)*. Mutation is applied afterwards followed by a *2-repair*, a variant of *2-opt local search* focusing on newly introduced edges.

The MPX proposed in [232, 122] has similarities with the traditional two–point crossover. To construct an offspring tour, a sub-path between two randomly chosen crossover points is copied from the first parent to the offspring. The partial tour is extended by copying edges from the second parent afterwards. If the the sub-path cannot be extended this way to retain a feasible solution, the edges from the first parent are checked. If there is no such edge from the first parent that can be used to extend the tour, a previously unvisited city is added from the second parent which comes next after the end point in the string. The table below shows an example.

MPX Crossover										
Parent A	4	2	**0**	**8**	**9**	**5**	3	7	6	1
Parent B	2	9	8	3	7	5	0	1	6	4
Offspring C	**0**	**8**	**9**	**5**	7	3	1	6	4	2

In the example, the highlighted sub-path from parent A is copied to the offspring. The offspring is extended by appending cities 7 and 3 so that the edges $(5, 7)$ and $(7, 3)$ contained in parent B are copied over. Edge $(3, 8)$ cannot be inserted since city 8 is already contained in offspring C. Looking at parent A, we see that both edges $(3, 5)$ and $(3, 7)$ cannot be used to extend the tour further. Hence, the city next to city 3 in parent B is identified: city 1. After adding city 1 two the partial offspring, the algorithm proceeds by inserting the remaining edges from parent B: edges $(1, 6)$, $(6, 4)$ and $(4, 2)$. The edge from the last to the first node is also contained in tour A, so we got only one foreign edge in offspring C.

Initially, a slightly different crossover had been used in ASPARAGOS [121] that is identical to the order crossover operator [69] except that a sub-path of the second parent is inverted before crossing over. In the literature, this operator has been called *Schleuter crossover* [204, 83] to avoid confusion with the MPX described above.

A major difference to other EAs is that the algorithm is asynchronous and parallel. In contrast to traditional genetic algorithms, there is no discrete generation model, i.e., there are no well distinguished (time-stepped) generations. Instead, selection for variation (mating selection) and selection for survival (replacement) are performed asynchronously. Furthermore, the population is spatially structured and consists of overlapping demes (local subpopulations). Mating (recombination) happens only between individuals within a deme (neighborhood). Therefore, no central control is needed and only local interactions occurs.

Thus, the algorithm is robust, is well suited for small populations and can be executed on parallel hardware. The term PGA [233] is often used for such a model with population structures.

Fine-grained PGAs for the TSP have also been studied in [155] and a variant of ASPARAGOS has been proposed in [40] called the *insular genetic algorithm*. A modified version of ASPARAGOS has been proposed in [123] called ASPARAGOS96 with a hierarchical population structure and slightly modified MPX and mutation.

Local Search and Recombination

Whitley et al. devised in [311] a crossover operator called *edge recombination operator*. This operator explicitly tries to keep the number of foreign edges in the child to a minimum. The operator itself has evolved over time. The first variant is the *enhanced edge recombination operator* [283] also known as *Edge-2* [204]. Further improvements have been made in *Edge-2* resulting in *Edge-3 operator* described in [204] and finally *Edge-4* [83] has been proposed. This family of operators is superior to MPX in a GA without local search; as shown in [204, 83], the *Edge* family has a smaller *failure rate* (number of introduced foreign edges) than MPX. But when local search is added to the algorithm, the picture changes and MPX becomes superior to the *Edge* operators. As with tour construction heuristics in combination with local search, in the case of evolutionary variation operators not necessarily the best *stand alone* operator performs best in combination with local improvement [204, 83].

Genetic Local Search

The name *Genetic Local Search* (GLS) was first used by Ulder et al. [301] to describe a evolutionary algorithm with recombination and consequently applied local search. Within this scheme all individuals of the population represent local optima with respect to the chosen local search. In [301], the population model of a GA has been used instead of a model with structured a population with asynchronous application of the variation operators. The recombination operator used was MPX, and opposed to *2-repair*, *2-opt* and the *LK* local search were incorporated.

In [44], Bui and Moon also propose a genetic local search algorithm with LK as the local search procedure. They developed a k-point crossover operator with additional repair mechanism to produce feasible offspring.

The approach described in this thesis as published in [105, 106, 210] is also a GLS and uses LK local search and a new recombination operator called DPX. This algorithm has won the first international contest on evolutionary optimization (ICEO) at the IEEE International Conference on Evolutionary Optimization [31, 105].

Walters [305] developed a two-point crossover for a nearest neighbor representation and a repair mechanism called *directed edge repair (DER)* to achieve feasibility of the solutions. He uses *3-opt* local search to improve the solutions further. Brood selection is incorporated to select the best of the children produced by crossover.

Katayama and Narihisa [165] proposed an EA with LK and small populations (just two individuals) and a heuristic recombination scheme. Their approach is similar to the iterated Lin-Kernighan heuristic but additional diversification is achieved by the recombination of the current solution and the best solution found. The results presented for large instances are impressive.

Other Highly Effective Evolutionary Algorithms

There are some other highly effective evolutionary algorithms for the TSP which do not belong to the class of memetic algorithms and are therefore worth mentioning.

Nagata and Kobayashi [234] devise an evolutionary algorithm that uses the *edge assembly crossover* to produce offspring. In this recombination operator, children are constructed by first creating an edge set from the edges contained in the parents (*E-set*) and than producing intermediate children for which the sub-tour constraint is generally not fulfilled. In order to obtain feasible offspring, sub-tours are merged in a greedy fashion based on the minimum spanning tree defined by the disjoint sub-tours.

Tao and Michalewicz [299] propose an evolutionary algorithm which is very easy to implement. The operator used in the algorithm is called *inver-over* since it can be regarded as a mixture of inversion and crossover. The operator has resemblance with the LK heuristic since a variable number of edges is exchanged. Thus, it is more a local search utilizing a population of solutions than an EA utilizing local search.

Möbius et al. [225, 224] propose a physically inspired method for the TSP called *thermal cycling with iterative partial transcription (IPT)*. To a population of solutions called “archive”, a heating phase (similar to simulated annealing with nonzero temperature) and a quenching phase (local search) is repeatedly applied. After quenching, IPT is used to improve the solutions in the archive further. IPT can be regarded as a form of recombination in which some of the alleles of one parent are copied to the other explicitly maximizing the fitness of the resulting individual.

Several other approaches have been published for solving the TSP. However, only few of them are suited for solving large TSP instances ($>> 1000$ cities) like the ones discussed here. It is meaningless to test an approach on just small TSP instances, since (a) there are exact methods for solving small instances to optimality in a few seconds, (b) simple local search algorithms are much faster than most EAs and produce comparable or better results, and (c) the behavior of an algorithm on small instances can not be used to conclude its behavior on large instances.

7.3 The Fitness Landscape of the TSP

Several researchers have studied the fitness landscape of the traveling salesman problem to find more effective search techniques. Even a theoretical analysis exists that coincides with conclusions drawn from experiments.

7.3.1 Distances between TSP tours

Besides landscape analysis, distance functions for solution vectors of optimization problems are important in a number of EA techniques, such as mechanisms for preventing premature convergence [86], or the identification of multiple solutions to a given problem [263]. Furthermore, they can be used to observe the dynamic behavior of the EA (or CCVA [87]) and to guide the search of the EA [105].

A suitable distance measure for TSP tours appears to be a function that counts the number of edges different in both tours: Since the fitness of a TSP tour is determined by the sum of the weights of the edges the tour consists of, the distance between two tours t_1 and t_2 can be defined as the number of edges in which one tour differs from the other. Hence

$$d(t_1, t_2) = |\{e \in E \mid e \in t_1 \wedge e \notin t_2\}|. \tag{7.1}$$

This distance measure has been used by several researchers, including [233, 36, 196, 106]. Recently, it has been shown that this distance function satisfies all four metric axioms [264].

Alternatively, a distance measure could be defined by counting the number of applications of a neighborhood search move to obtain one solution from the other. In the case of the *2-opt* move, the corresponding distance metric d_{2-opt} is bound by $d \leq d_{2-opt} \leq 2d$ [196].

With this distance measure, the neighborhoods based on edge exchange can be defined as

$$\mathcal{N}_{k\text{-}opt}(t) = \{t' \in T : d(t, t') \leq k\}, \tag{7.2}$$

with T denoting the set of all tours of a given TSP instance. Note that the node exchange neighborhood is a small subset of the *4-opt* neighborhood and the node (re)insertion neighborhood is a subset of the *3-opt* neighborhood since 4 edges and 3 edges are exchanged, respectively.

7.3.2 Autocorrelation Analysis

Stadler and Schnabl [281] performed a landscape analysis of random TSP landscapes considering different neighborhoods: the *2-opt* and the *node exchange* neighborhood. Their results can be summarized as follows.

For the symmetric TSP, both landscapes (based on *2-opt* and *node exchange*) are AR(1) landscapes. The random walk correlation function for random landscapes is of the form

$$r(s) \approx \exp(-s/\ell) = \exp(-b/n \cdot s), \tag{7.3}$$

with n denoting the number of nodes/cities of the problem instance and b denoting the number of edges exchanged between neighboring solutions. Thus, for the *2-opt* landscape, the normalized correlation length $\xi = \ell/n$ is $\frac{1}{2}$, for the *node re-insertion* landscape ξ is $\frac{1}{3}$, and for the *node exchange* landscape ξ is $\frac{1}{4}$. This result coincides with experimentally obtained results that *2-opt* local search is much more effective than local search based on *node exchange* or *node re-insertion* [261]. The formula 7.3 implies that a landscape with a strict *3-opt* neighborhood is more rugged than a landscape with a *2-opt* neighborhood. One may conclude that a *2-opt* local search performs better than a *3-opt* local search. However, the opposite is true, since the *3-opt* neighborhood is much greater than the *2-opt* neighborhood and the *3-opt* neighborhood as defined above contains the *2-opt* neighborhood. Therefore, a *3-opt* local search cannot perform worse than a *2-opt* local search in terms of solution quality.

In case of the asymmetric TSP, the above equation holds, too, with the exception that there is no *2-opt* move if the distance matrix is asymmetric. Reversing a sub-path in a ATSP tour leads generally to a k-change depending on the length of the sub-path. Stadler and Schnabl [281] have shown that such reversals yield a random walk correlation function of the form

$$r(s) \approx \frac{1}{2}\delta_{0,s} + \frac{1}{2}\exp(-4s/n), \tag{7.4}$$

where δ denotes the Dirac function. $\delta_{0,s}$ is defined as

$$\delta_{0,s} = \begin{cases} 1, & \text{if} \quad s = 0 \\ 0, & \text{otherwise} \end{cases} \tag{7.5}$$

in the discrete case. The Dirac function plays an important role in signal theory.

7.3.3 Fitness Distance Correlation Analysis

The correlation of fitness of local optima and distance to the optimum solution has already been studied by Boese [35, 34] in order to derive a suitable search strategy for the TSP. However, he concentrated in his studies [35] on a single TSP instance contained in TSPLIB [260], a public accessible library of TSP instances.

To obtain more general information, additional instances have been analyzed for which the results are presented in the following. The instances are selected to cover different problem sizes as well as problem characteristics. The first three instances denoted mpeano7, mnpeano7, and David5 are fractal instances based on L-systems (such as the Koch-curve) with known optimum tours described in [229, 228, 238]. The number in the name denotes the order of the fractal.

The other nine instances are chosen from TSPLIB. The first instance denoted ts225 is known to be hard to solve exactly by Branch-&-Cut algorithms [10] although it has a small number of cities. Instance pcb442 is a PCB production instance with a regular location of the nodes. The instances att532, pr1002, and pr2392 are instances derived from real city locations. rat783 is an instance with random distribution of the cities in a rectangular area. dsj1000 denotes an instance with clustered cities. And finally, the instances fl1400 and fl1577 are PCB drilling problems. The latter of the two has been the smallest unsolved problem in TSPLIB for a long time. Recently, it could be solved to optimality, however. In Figure 7.4, some characteristic instances are displayed.

To obtain insight into the structure of the fitness landscapes of these instances, experiments have been conducted in which the (cor-)relation of fitness and distance to the optimum of locally optimum solutions has been investigated. For instances with more than one known optimum solution, the distances to the nearest optimum was considered. For example, the number of optima found in experiments for the instances ts225, rat783, and fl1400, is 147, 17, and 7, respectively. For the first two instances, the average distance between the optima is 25.8 and 9.5, respectively. The optima found for instance fl1400 have an average distance of 336.6. It is assumed that all fl instances have a high number of global optima. Since just one global optimum was known to the author at the beginning of the experiments, no other global optima has been considered in the FDC.

In a first series of experiments, the local optima were produced by a *fast 3-opt* local search applied to randomly generated solutions. The results are presented in Table 7.1. In the first column, the name of the instance is displayed, and in the second column the problem size n is given. In columns three through seven, the minimum distance of the local optima to a global optimum ($\min d_{opt}$), the average distance of the local optima to the global optimum ($\overline{d}_{opt}$), the average distance between the local optima ($\overline{d}_{loc}$), the number of distinct local optima (N_{3-opt}) out of 2500, and the fitness distance correlation coefficient (ϱ) are provided, respectively. Additionally, the normalized average distance, i.e., the average distance of the local optima to the global optimum divided by the maximum distance in the search space n is shown in column four in parentheses. In case of more than one known global optimum, the distance to optimum means the distance to the nearest optimum.

In case of the fractal instances mpeano7 and David5, the optimum could be found with fast 3-opt. The average distance to the optimum is very small compared to the maximum distance in the search space, and the locally optimum solutions are close together. mpeano7 appears to have a small number of local optima since in the analysis only 840 distinct local optima could be found. For the problems contained in TSPLIB, the normalized average

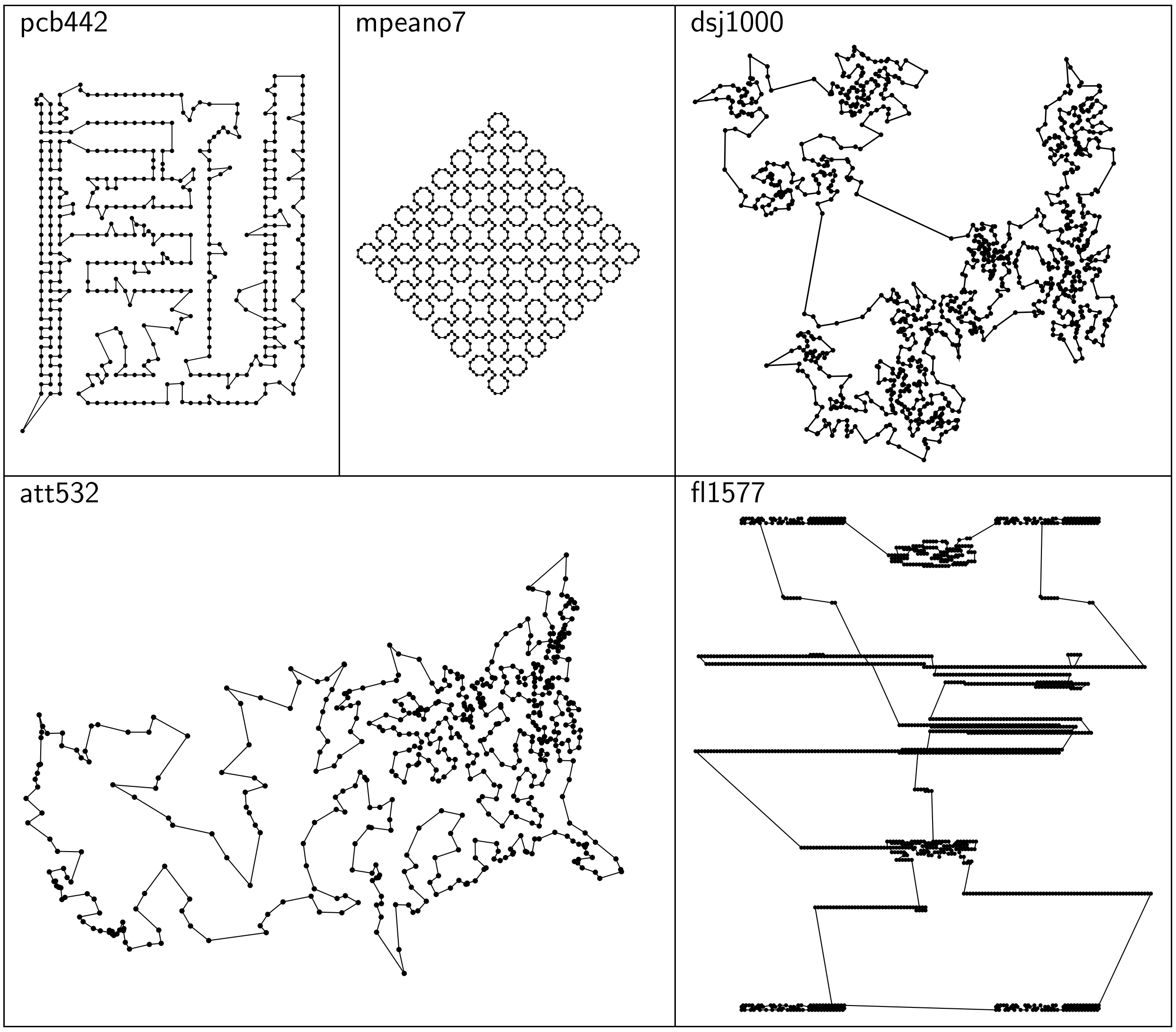


Figure 7.4: Optimum tours of five TSP instances

distance to the optimum is about 0.2 with one exception: for fl1400 the value is about 0.4. Thus, all TSPLIB instances have local optima with a significantly higher distance to the optimum than the fractal instances. For all instances, the average distances between the local optima are similar to the average distance to the optimum. The correlation coefficient is high for the instances based on real city instances and clusters seem to affect correlation negatively. For the random instance rat783, the correlation coefficient is highest, and it is lowest for the drilling problems and ts225.

For the same set of instances, a second series of experiments has been conducted. In these experiments, the local optima were generated with the Lin-Kernighan heuristic rather than with 3-opt. The results are displayed in Table 7.2. The local optima generated by the LK heuristic show the same properties than those obtained by 3-opt. The correlation coefficients are slightly higher for almost all TSPLIB instances, and in case of the fractal instances they are close to 1. Fitness distance plots for some of the instances are provided in Figure 7.5. The distance to the optimum is plotted against the fitness (cost) difference between the locally optimum solutions and the fitness of the global optimum ($\Delta f = c(\pi_{loc}) - c(\pi_{opt})$). The

Table 7.1: Results of the Fitness Distance Analysis for 3-opt Solutions of the TSP

Instance	n	$\min d_{opt}$	$\overline{d}_{opt}$	$\overline{d}_{loc}$	N_{3-opt}	ϱ
mnpeano7	724	20	85.32 (0.12)	138.53	2500	0.50
mpeano7	852	0	1.93 (<0.01)	3.83	840	0.40
David5	1458	0	29.98 (0.02)	57.55	2498	0.56
ts225	225	19	33.90 (0.15)	35.07	2496	0.18
pcb442	442	63	105.95 (0.24)	109.74	2500	0.48
att532	532	36	106.48 (0.20)	123.17	2500	0.57
rat783	783	83	151.82 (0.19)	184.77	2500	0.68
dsj1000	1000	122	207.93 (0.21)	239.87	2500	0.36
pr1002	1002	123	203.00 (0.20)	242.16	2500	0.57
fl1400	1400	504	574.85 (0.41)	561.26	2500	0.07
fl1577	1577	152	239.90 (0.15)	260.10	2500	0.27
pr2392	2392	283	430.04 (0.18)	496.62	2500	0.63

Table 7.2: Results of the Fitness Distance Analysis for LK Solutions of the TSP

Instance	n	$\min d_{opt}$	$\overline{d}_{opt}$	$\overline{d}_{loc}$	N_{LK}	ϱ
mnpeano7	724	0	20.94 (0.03)	39.09	118	0.99
mpeano7	852	0	13.56 (0.02)	25.99	87	0.99
David5	1458	0	3.82 (<0.01)	7.55	137	0.94
ts225	225	20	33.60 (0.15)	34.98	2497	0.21
pcb442	442	61	105.92 (0.24)	109.82	2500	0.50
att532	532	47	106.29 (0.20)	122.71	2500	0.54
rat783	783	75	151.38 (0.19)	184.51	2500	0.70
dsj1000	1000	105	208.19 (0.21)	240.01	2500	0.36
pr1002	1002	108	202.15 (0.20)	241.77	2500	0.60
fl1400	1400	511	575.23 (0.41)	560.71	2500	0.06
fl1577	1577	151	238.95 (0.15)	259.55	2500	0.34
pr2392	2392	310	429.35 (0.18)	496.47	2500	0.64

instance mpeano7 shows perfect correlation between the fitness difference and the distance to the optimum. The local optima form a straight line originating from the optimum. The plot for ts225 looks quite different: for some fitness values, there are several local optima while for most fitness values there is not even a single one leading to large gaps in fitness of the local optima. Problems att532, rat783, and pr2392 exhibit a "cloud" of local optima in their scatter plots. The mean of the points are oriented along a straight line. The clustered instance dsj1000 is similar but there is no orientation towards the optimum. This phenomenon becomes more apparent in the problems fl1400 and fl1577. The mean of the points are distributed parallel to the δf-axis.

The analysis has shown that local optima in the TSP are found in a small region of the search space: on average, more than $\frac{3}{4}$ of the edges are common to all local optima with

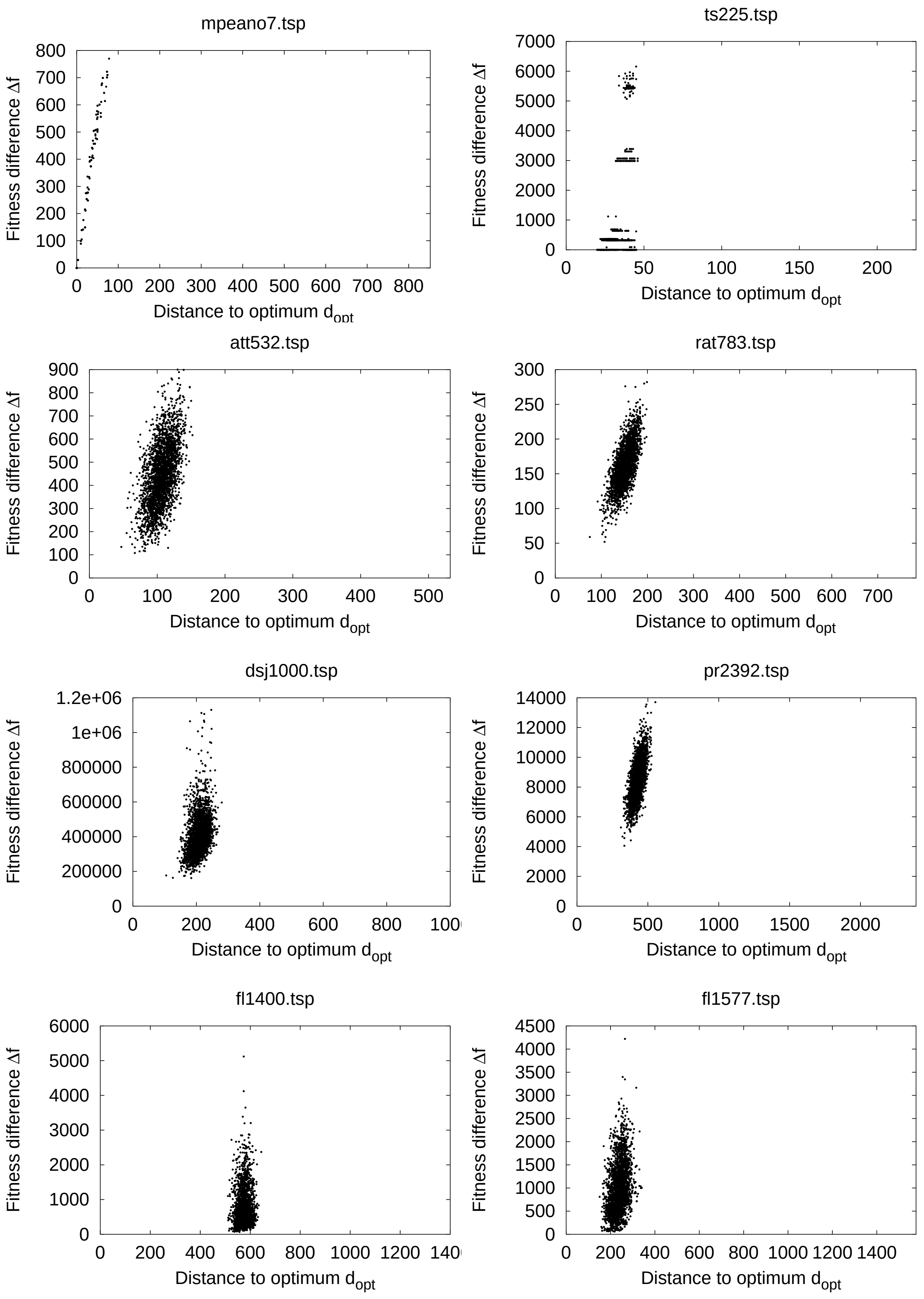


Figure 7.5: Fitness-Distance Plots produced with LK local Search

one exception (fl1400). Furthermore, fitness and distance to the optimum are correlated for most instances, and the average distance between the local optima is similar to the distance to the optimum. Thus, the global optimum appears to be more or less central among the local optima. Boese calls the structure of the TSP landscape the *big valley structure*, since local optima are closer together if they are closer to the optimum, and the smaller the cost, the closer they are to the optimum. However, the analysis has also shown that not all instances exhibit this structure as, for example, ts225. Furthermore, the analysis indicates that problems from application domains such as the drilling problems are harder to solve than randomly generated instances with uniform distribution. The fractal instances on the other hand are very easy to solve. They are not well suited as benchmark problems for highly effective heuristics, since they do not have same the characteristics as the instances arising in TSP applications. The big valley structure can be well exploited by a memetic algorithm with recombination since good solutions are more likely to be found near other local optima and most recombination operators produce solutions that lie “between” other solutions (respectful recombination). Furthermore, an evolutionary algorithm usually increases fitness of the solutions contained in the population while simultaneously decreasing the distance between the solutions.

7.4 A Memetic Algorithm for the TSP

The memetic algorithm for the TSP is as outlined in the previous chapter: a population of local optimum solutions is evolved over time by applying evolutionary variation operators (mutation and recombination operators). To ensure that the individuals in the population are local optima, after each application an evolutionary variation operator, local search is applied. This includes the initialization phase of the population in which solutions are constructed from scratch: A local search procedure is applied to these solutions so that even the first generation consists exclusively of local optima.

The problem-specific parts of the algorithm comprise initialization, local search, and the evolutionary variation operators.

7.4.1 Initialization and Local Search

To initialize the population of the MA, a local search procedure is applied to solutions constructed by the randomized greedy heuristic described above. However, the randomization technique proposed by Johnson *et al.* [158], is not well suited for initialization of an MA since the resulting solutions are very similar. Therefore, a variant is used: Before the greedy construction scheme is applied, $\frac{n}{4}$ edges are inserted in the tour solution randomly by selecting the edge to the nearest or second nearest unvisited neighbor of a randomly chosen unvisited city. The edge to the nearest city is selected with a probability of 0.66 and the edge to the second nearest city is selected with probability 0.33. After an edge has been inserted, the endpoints of the edge are marked as visited to guarantee that the partial solution will not become an infeasible solution.

Since the Lin-Kernighan heuristic is the best local search heuristic proposed for the TSP, it is used in our algorithm. In some cases, the simpler fast 3-opt heuristic is used when it is more efficient to use a fast but less elaborate local search.

7.4.2 Variation Operators

Mutation operators used in simple evolutionary algorithms are not suited for use in MAs, since subsequently applied local search procedures will usually revert the changes made. For example, the inversion operator randomly exchanging two edges is ineffective when 2-opt, 3-opt or LK local search is used. Therefore, in MAs alternative mutation operators are required.

The Mutation Operator

The mutation operators of our algorithms are based on edge exchange. There a two variants, one of which produces arbitrary exchanges of a predefined number of k edges, and and the other one which produces non-sequential edge exchanges. The smallest of such an exchange is displayed in Figure 7.4.2 and involves four edges [192]. It stands in contrast to the sequential edge exchanges performed by the Lin-Kernighan heuristic as described above. Since the LK heuristic performs sequential changes, the probability is minimized that LK

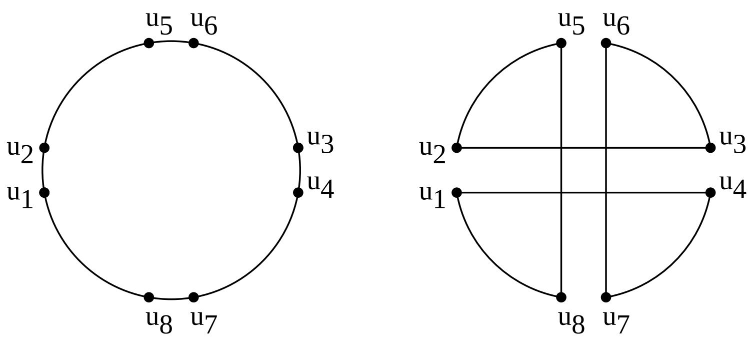


Figure 7.6: The non-sequential 4-change

reverses mutation if non-sequential edge exchanges are utilized. In the effective *iterated LK heuristic*, the non-sequential four-change is used as a mutation operator to escape from the basins of attraction of local optima.

The DPX Recombination Operator

In case of recombination, previously published operators for EAs without LS can be used in MAs, but as shown in [204, 83], there may be others that are better suited for the use in MAs. These operators may be ineffective when used without LS.

The *distance preserving crossover (DPX)* proposed in [106, 105] is such an operator that is only useful in combination with local search. In contrast to other recombination operators such as the edge recombination operators [311, 283], it forces the inclusion of foreign edges in the offspring instead of preventing it.

DPX tries to generate an offspring that has equal distance to both of its parents, i.e., its aim is to achieve that the three distances between offspring and parent 1, offspring and parent 2, and parent 1 and parent 2 are identical. It works as follows: the content of the first parent is copied to the offspring and all edges that are not in common with the other parent are deleted. The resulting parts of the broken tour are reconnected without using the non-shared edges of the parents. A greedy reconnection procedure is employed to achieve this: if the edge (i, j) has been destroyed, the nearest available neighbor k of i among the remaining tour fragments is taken and the edge (i, k) is added to the tour, provided that (i, k) is not contained in the two parents. In order to illustrate the DPX operator, let us consider an example.

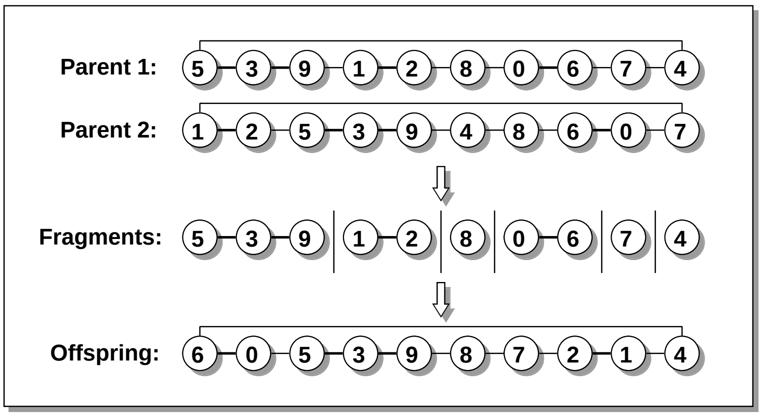


Figure 7.7: The DPX recombination operator for the TSP

Suppose that the two parents shown in Figure 7.4.2 are given, then copying parent 1 to the offspring and deleting the edges not contained in both parents leads to the tour fragments 5 3 9 - 1 2 - 8 - 0 6 - 7 - 4. The greedy reconnection procedure fixes the broken connections by producing the offspring shown in Figure 7.4.2 as follows. First, a city is chosen randomly as the starting point for the reconnection. Let us assume that the city to begin with is city 6, then the other endpoint (city 0) of the fragment containing city 6 is considered and its nearest neighbor in the set of available cities, {5,9,1,2,4}, is determined. The set of available cities only contains the start and endpoints of not yet visited tour fragments. City 8 and city 7 are not contained in this set, because it is not desirable to reinsert edge $(0, 8)$ or edge $(0, 7)$, since they are contained in parent 1 or parent 2, respectively. Let us assume that in the example the nearest neighbor to city 0 is city 5, so city 0 is connected to city 5, and the end of the connected fragment (city 9) is considered. At this point the set of available cities is {2,8,7}. The procedure is repeated until all fragments have been reconnected. Note that the distance d between the offspring and both parent 1 and parent 2 is identical to the distance between the two parents ($d = 6$), hence the name distance preserving crossover.

In some rare cases, it is necessary to introduce backtracking into the greedy reconnection procedure to fulfill the distance criterion. E.g., if in the example above the edges $(2, 0)$, $(6, 4)$, $(7, 8)$, $(7, 9)$, and edge $(1, 8)$ are inserted due to the nearest neighbor relations, the remaining edge to close up the tour is edge $(4, 5)$. Since this edge is contained in parent A, the resulting child will will not fulfill the distance criterion: the distance to parent A becomes 5 and the distance to parent B becomes 6. In such a situation, a backtracking mechanism trying alternative edges in preceding steps has to be employed. However, in the MA used in the experiments, the DPX operator does not include backtracking since it is not worth to spend the extra computation time for backtracking. The cases for which the distance criterion is not obeyed are extremely rare and it has been shown that the use of backtracking in the DPX has no influence on the overall performance of the MA. Thus, the easier to implement "one-pass" DPX is used in the MA experiments.

The Generic Greedy Recombination Operator

Another recombination operator has been developed that utilizes the greedy construction scheme of the greedy heuristic described above. The *generic greedy recombination operator (GX)* consists of four phases. In the first phase, some or all edges contained in both parents are copied to the offspring tour. In the second phase, new short edges are added to the offspring that are not contained in one of the parents. These edges are selected randomly among the shortest edges emanating from each node. These edges are with high probability contained in (near) optimum solutions and are thus good candidates for edges in improved tours. In a third phase, edges are copied from the parents by making greedy choices. Here, edges may be included that are not common to both of the parents. Edges are inserted in order of increasing length, and only candidate edges are considered, i.e., edges that violate the TSP constraints. In the fourth and last phase, further edges are included utilizing the greedy construction scheme of the greedy heuristic described above until the child consists of n edges and is thus a feasible TSP tour. The pseudo code of the recombination operator is provided in Figure 7.8.

The GX operator has three parameters: the common edges inheritance rate (cRate) that determines the probability that a common edges is added to the child and is thus a control parameter for the first phase. With a rate of 1.0, respectful recombination is achieved, all other rates lead to unrespectful recombination. The second phase is controlled by the new edges introduction rate (nRate) that determines the number of new edges to insert. A rate of 0.5, for example, determines that half of the remaining edges to insert after phase one are new edges that are short but not contained in one of the parent solutions. The number of edges to inherit from the parents including edges not common two both parents is determined by the inheritance rate (iRate). In the last phase, edges in increasing length are chosen that may or may not be found in the parents.

Local Search and Recombination

In an MA, recombination operators are desired that are efficiently in combination with the local search that is applied after a child has been generated. Thus, it makes sense to tune the local search for its operation after recombination. The landscape analysis has shown that there is correlation between tour length and distance to the optimum of local minima and that a local optimum with high fitness (short tour length) is contained near other local optima with high fitness. Therefore, it makes sense to restrict a local search after recombination to search only the region around or between the two parents. This can be accomplished by fixing all common edges that have been included in the child in the first step of recombination. The edges that are common to both parents can be regarded as the “building blocks” of the evolutionary search and should be found in good offspring tours. Fixing these edges prevents the local search to replace these edges by others and reduces the running time of the local search considerably. From the perspective of the local search, the fixing of edges reduces the problem size since fixed edges are never considered during the search for edge exchanges.

The landscape analysis has shown that less than one fourth of the edges in the local optima are different. Thus, in the first generation of an MA, a local search operates on a problem with a dimensionality of one fourth of the original one if the fixing of edges is performed during recombination. Since with ongoing evolution, the distance between the members of the population diminishes, the size of the problem becomes smaller for the local

```
procedure GreedyRecombination(a,b in X; cRate, nRate, iRate: Real): X;

  begin
    let x be the edge set of the partial offspring tour;
    let remaining = n;
    /* Copy common edges */
    foreach edge e in a do
      if (e in a and cRate < random[0,1)) then
        add e to x;
        remaining := remaining − 1;
      end;
    end;
    /* Insert new edges */
    for k:=1 to (remaining ∗ nRate) do
       i := n ∗ random[0,1);
       j := select from (the 5 nearest neighbors of i)
            with (i,j) feasible and (i,j) not in a or b;
       add edge (i,j) to x;
       remaining := remaining − 1;
    end;
    /* Inherit edges from parents */
    for k:=1 to (remaining ∗ iRate) do
      parent := select randomly from (parent a, parent b);
      if (parent has a candidate edges) then
        edge := select from (shortest, second shortest candidate edge);
        add edge to x;
        remaining := remaining − 1;
      end;
    end;
    /* greedy completion */
    while (remaining > 0) do
      edge := select from (shortest, second shortest candidate edge);
      add edge to x;
      remaining := remaining − 1;
    end;
  end;
```

Figure 7.8: The Generic Greedy Recombination Operator

search in each generation. This leads to a radically reduced running time for the local search.

7.4.3 Implementation Details

In the implementation of the algorithms for the TSP described in this chapter, a nearest neighbor list of size $m = 100$ for each node is maintained, which is initialized by nearest neighbor queries on a two-dimensional binary search tree [28]. In the local search procedures, a data structure for maintaining don't look bits is incorporated, with the local search for the initial population starting with all don't look bits set to zero. After recombination has been performed, only the don't look bits of the nodes that are incident to the edges not shared by both parents are cleared. Similarly, after mutation, only nodes incident to the edges newly included in the tour have their don't look flags set to zero. This focuses the search of the hill-climber to the promising regions of the search space and also reduces the time for checking the interesting members of the neighborhood.

Additionally, in the algorithm for the TSP, data structures have been incorporated to deal with large instances of up to 100.000 cities. Since for large instances it is not possible to store the entire distance matrix in main memory, the Euclidean distances are computed online. This is a rather expensive operation, so a distance cache of size $3 \cdot n$ is maintained, where the first n entries are used to cache the distances of the edges in the current tour and the remaining $2 \cdot n$ entries are organized as described in [28]. The average hit rate of the cache varies between 80% and 95%.

Another target for optimizations is the Lin-Kernighan heuristic itself. Most of the computation time is spent in submoves that will be reversed later in the algorithm. Hence, it is profitable to distinguish between tentative and permanent moves. Applegate and Cook have proposed a *segment tree* data structure for efficiently managing tentative moves, as described in [104]. Instead of using a segment tree, the algorithms described here operate on a segment list that represents a tentative tour. Operations performing a flip on this tentative tour are highly optimized, such that a high performance gain compared to the simple array representation can be achieved. The running times for all operations are in $O(1)$, since the data structure is limited to perform 20 flips only. In practice, this has been proven to be sufficient.

7.4.4 Performance Evaluation

Several experiments have been conducted to evaluate the performance of memetic algorithms for the TSP. All experiments described in the following were conducted on a PC with Pentium III Processor (500 MHz) under Linux 2.2. All algorithms were implemented in *C++*.

In a first set of experiments, several recombination operators for the TSP were tested under the same conditions on three selected TSP instances contained in TSPLIB: att532, pr1002, and fl1577. To get a clear picture of the operator effectiveness, no additional mutation was performed and the restart mechanism was disabled during the runs. Furthermore, a fast *2-opt* local search was used in the MAs that is not as effective as *3-opt* local search or the Lin-Kernighan heuristic to reduce the strong influence of the (sophisticated) local search. The recombination operators MPX, DPX, and the generic greedy recombination operator was studied with various parameter settings. The population was set to $P = 100$ in all runs, and the variation operator application rate was set to 0.5, i.e., 50 offspring were generated per generation. The results of the experiments are summarized in Table 7.3. For each instance/operator, the average number of generations, the shortest tour length found, and the percentage access over the optimum solution value is provided. For the GX operator, the values for *cRate*, *nRate* and *iRate* are provided in the form *cRate/nRate/iRate*. For example, a parameter setting of 1/0.25/0.75 means that the common inheritance rate *cRate* was set to 1.0, the new edges introduction rate *nRate* was set to 0.25, and the inheritance rate *iRate* was set to 0.75. The dot in each column block indicates the best result within this block.

For all three instances, MPX and DPX are outperformed by GX for some of the parameter

Table 7.3: Comparison of MA Recombination Strategies for the TSP (2-opt)

Operator	att532		pr1002		fl1577	
DPX	1565	27793.0 - 0.386%	664	266240.5 - 2.778%	653	22314.0 - 0.292%
MPX	2691	27772.0 - 0.311%	3404	261695.5 - 1.023%	1240	22347.8 - 0.444%
GX-Params						
1/1/1	650	27738.7 - 0.190%	307	268183.5 - 3.528%	554	22295.6 - 0.210%
1/1/0.75	708	27744.7 - 0.212%	354	268072.9 - 3.485%	592	22306.7 - 0.259%
1/1/0.5	725	27740.0 - 0.195%	415	267033.1 - 3.084%	585	22304.0 - 0.247%
1/1/0.25	669	27772.0 - 0.311%	304	268487.4 - 3.645%	580	22296.5 - 0.213%
1/0.5/1	868	27729.8 - 0.158%	759	260907.8 - 0.719%	624	22294.8 - 0.206%
1/0.5/0.75	929	27727.0 - 0.148%	733	261981.0 - 1.133%	713	22294.6 - 0.205%
1/0.5/0.5	923	27725.2 - 0.142%	808	261121.2 - 0.801%	682	22296.7 - 0.214%
1/0.5/0.25	892	27723.9 - 0.137%	832	260723.4 - 0.648%	641	22303.5 - 0.245%
1/0.25/0	928	27724.5 - 0.139%	1223	260671.2 - 0.628%	690	22304.5 - 0.250%
1/0.25/0.75	1091	• 27719.2 - 0.120%	1430	260683.9 - 0.633%	769	22294.8 - 0.206%
1/0.25/0.5	1065	27722.4 - 0.131%	1422	260585.9 - 0.595%	684	22311.7 - 0.282%
1/0.25/0.25	998	27723.3 - 0.135%	1334	• 260508.6 - 0.565%	696	22307.0 - 0.261%
1/0/1	956	27763.5 - 0.280%	1321	261379.9 - 0.901%	736	22323.4 - 0.335%
1/0/0.75	1071	27728.0 - 0.152%	1481	260894.8 - 0.714%	735	• 22287.8 - 0.174%
1/0/0.5	1035	27725.4 - 0.142%	1434	260949.5 - 0.735%	744	22312.0 - 0.283%
1/0/0.25	1006	27737.7 - 0.186%	1412	260984.0 - 0.749%	719	22326.2 - 0.347%
0.75/0.5/1	201	28429.8 - 2.686%	226	269423.5 - 4.007%	212	22725.8 - 2.143%
0.75/0.5/0.75	224	28435.5 - 2.707%	254	269423.5 - 4.007%	230	22725.8 - 2.143%
0.75/0.5/0.5	215	28435.5 - 2.707%	243	269423.5 - 4.007%	225	22725.8 - 2.143%
0.75/0.5/0.25	206	28434.8 - 2.705%	232	269423.5 - 4.007%	219	22725.8 - 2.143%
0.75/0.25/0	233	27986.0 - 1.084%	229	269271.2 - 3.948%	227	22679.0 - 1.932%
0.75/0.25/0.75	269	28230.8 - 1.968%	288	269423.5 - 4.007%	269	22671.2 - 1.897%
0.75/0.25/0.5	254	28063.3 - 1.363%	258	269335.2 - 3.972%	254	22657.9 - 1.838%
0.75/0.25/0.25	243	27976.5 - 1.049%	240	269384.7 - 3.991%	239	22649.5 - 1.800%
0.75/0/1	407	27869.0 - 0.661%	422	263536.0 - 1.734%	270	22583.3 - 1.503%
0.75/0/0.75	517	27771.5 - 0.309%	705	• 261696.8 - 1.024%	620	• 22319.3 - 0.316%
0.75/0/0.5	457	• 27747.2 - 0.221%	558	262236.0 - 1.232%	398	22415.2 - 0.747%
0.75/0/0.25	415	27750.5 - 0.233%	435	262634.5 - 1.386%	298	22492.2 - 1.093%
0.5/0.25/0	156	28394.2 - 2.558%	179	269400.0 - 3.998%	161	22725.8 - 2.143%
0.5/0.25/0.75	191	28433.2 - 2.699%	224	269423.5 - 4.007%	187	22725.8 - 2.143%
0.5/0.25/0.5	172	28414.0 - 2.630%	201	269423.5 - 4.007%	178	22724.8 - 2.139%
0.5/0.25/0.25	162	28373.5 - 2.483%	187	269423.5 - 4.007%	170	22725.8 - 2.143%
0.5/0/1	195	28041.8 - 1.285%	216	266696.7 - 2.954%	174	22693.8 - 1.999%
0.5/0/0.75	403	27870.7 - 0.667%	455	• 263020.8 - 1.535%	363	• 22416.0 - 0.751%
0.5/0/0.5	293	• 27838.5 - 0.551%	316	263258.8 - 1.627%	242	22530.1 - 1.263%
0.5/0/0.25	220	27894.7 - 0.754%	227	265673.8 - 2.559%	192	22628.6 - 1.706%
ILS	61365	27777.7 - 0.331%	126457	260683.6 - 0.633%	150797	22369.2 - 0.540%
NS4	744	27860.2 - 0.629%	1438	261922.0 - 1.111%	1633	22304.0 - 0.247%
Time:		60 sec.		120 sec.		200 sec.

settings: all GX variants with a common inheritance rate of 1.0 and a new edge introduction rate of 0.25 perform better than MPX and DPX. However, the best parameter setting for GX is for each of the instances a different one implying that there is no “golden rule” leading to the best recombination strategy for all TSP instances! For example, the best setting for fl1577 is 1/0/0.75 but all other combinations with *nRate* set to 0.0 do not perform as good as the GX variants with *nRate* set to 0.25. Furthermore, it becomes apparent that respectfulness is a very important property of recombination operators since all GX versions with a common inheritance rate less than 1 perform significantly worse than the respectful greedy recombination operators. However, choosing a high inheritance rate can compensate the phenomenon to an extent since the common edges of the parents have a chance to be included in the offspring in the third phase of the generic recombination. Additionally, iterated *2-opt* local search (ILS) and a MA with the non-sequential four-change mutation (NS4) and no recombination has been applied to the three instances. The mutation based algorithms perform relatively good but can not compete with the greedy recombination MAs. The correlation structure of the landscape can be exploited by a recombination-based MA. For the instance fl1577, the MA with NS4 performs much better than ILS indicating that for such a type of landscape search from multiple points (population-based search) is more promising.

In the second experiment, the fast *2-opt* local search has been replaced by the Lin-Kernighan heuristic. The population size was set to 40, the variation operator application rate was set to 0.5, i.e., 20 offspring were generated per generation, and restarts were enabled with a diversification rate of 0.3. The results obtained from experiments with MAs using DPX, MPX, respectful GX, non-sequential-four-change mutation (denoted NS4) in comparison to the iterated Lin-Kernighan heuristic (ILK) are displayed in Table 7.4. For each instance/operator pair, the average number of generations, and the percentage access over the optimum solution value is provided. For the GX operator, the values for *nRate* and *iRate* are provided in the form *nRate/iRate*. *cRate* was set to 1.0 in all experiments. The dot in each row indicates the best result for an instance.

Here, the performance differences of the MAs are in most cases not significant. For the problems att532, rat783, and pr1002 all algorithms perform well with only small differences, except for the MA with MPX recombination in case of pr1002. Surprisingly, this MA performs significantly worse than the other algorithms. For *fl1577* the MAs with DPX and GX outperform all other competitors with the MA using DPX being the best. For *pr2392*, all recombination based algorithms perform similar, but the MA with mutation and ILK perform significantly worse. In case of *pcb3038*, the largest instance considered, all results lie close together. The MAs with DPX and MPX outperform ILK and the MA with NS4. In the greedy recombination GAs, high differences can be observed. The best results are obtained with a new edge introduction rate of 0.25. The results show no clear tendency, and often the values lie to close together to being significantly different. However, in none of the cases, ILK or the MA with mutation is able to outperform the MA using DPX or the best greedy recombination. The performance differences between mutation and recombination operators have become more apparent using *2-opt* local search. For larger instances, this may be also observed for MAs with the LK heuristic.

In an additional experiment, the combination of recombination and mutation operators in a MA has been investigated. In the same experimental setup as before, the MAs with DPX and MPX recombination have been run with the non-sequential four change mutation operator (NS4). The results are provided in Table 7.5. The table contains the results

Table 7.4: Comparison of MA Recombination Strategies for the TSP (LK)

	att532	rat783	pr1002	fl1577	pr2392	pcb3038
ILK	0.046 %	0.018 %	0.065 %	0.158 %	0.215 %	0.135 %
DPX	0.030 %	0.004 %	0.023 %	• 0.028 %	0.068 %	0.113 %
MPX	• 0.021 %	• 0.001 %	0.169 %	0.142 %	0.054 %	0.128 %
NS4	0.055 %	0.010 %	0.020 %	0.181 %	0.119 %	0.171 %
GX 1.0/1.0	0.030 %	0.007 %	0.036 %	0.055 %	0.042 %	0.132 %
GX 1.0/0.75	0.035 %	0.026 %	0.022 %	0.058 %	0.053 %	0.211 %
GX 1.0/0.5	0.040 %	0.008 %	0.011 %	0.045 %	0.050 %	0.171 %
GX 1.0/0.25	0.043 %	0.006 %	0.013 %	0.051 %	0.047 %	0.146 %
GX 0.5/0.5	0.033 %	0.006 %	0.009 %	0.042 %	0.037 %	0.112 %
GX 0.5/0.75	0.031 %	0.007 %	0.031 %	0.048 %	0.055 %	0.175 %
GX 0.5/0.5	0.035 %	0.008 %	0.005 %	0.046 %	0.051 %	0.143 %
GX 0.5/0.25	0.037 %	0.009 %	0.011 %	0.037 %	0.044 %	0.136 %
GX 0.25/0	0.026 %	0.002 %	0.017 %	0.044 %	0.022 %	0.125 %
GX 0.25/0.75	0.038 %	0.012 %	0.003 %	0.041 %	0.031 %	0.151 %
GX 0.25/0.5	0.035 %	0.006 %	0.002 %	0.036 %	0.025 %	0.111 %
GX 0.25/0.25	0.041 %	0.005 %	0.002 %	0.040 %	0.023 %	• 0.111 %
GX 0.0/1.0	0.045 %	0.008 %	0.006 %	0.052 %	• 0.020 %	0.123 %
GX 0.0/0.75	0.036 %	0.003 %	• 0.000 %	0.043 %	0.027 %	0.115 %
GX 0.0/0.5	0.034 %	0.011 %	0.008 %	0.052 %	0.029 %	0.122 %
GX 0.0/0.25	0.037 %	0.004 %	0.002 %	0.050 %	0.035 %	0.123 %
Time:	60 sec.	80 sec.	200 sec.	300 sec.	400 sec.	800 sec.

Table 7.5: Comparison of MAs with Recombination and Mutation (NS4) for the TSP

	att532	rat783	pr1002	fl1577	pr2392	pcb3038
DPX	0.030 %	0.004 %	0.023 %	0.028 %	0.068 %	0.113 %
DPX, $m = 0.1$	0.017 %	0.001 %	0.012 %	0.027 %	0.021 %	0.099 %
DPX, $m = 0.5$	0.017 %	0.007 %	0.000 %	0.041 %	0.043 %	0.106 %
MPX	0.021 %	0.001 %	0.169 %	0.142 %	0.054 %	0.128 %
MPX, $m = 0.1$	0.013 %	0.000 %	0.041 %	0.146 %	0.053 %	0.094 %
MPX, $m = 0.5$	0.025 %	0.005 %	0.054 %	0.138 %	0.047 %	0.103 %
Time:	60 sec.	80 sec.	200 sec.	300 sec.	400 sec.	800 sec.

achieved with DPX and MPX without mutation as well as the results for a mutation operator application rate of $m = 0.1$ and $m = 0.5$. The number of offspring per generation produced by mutation is $m \cdot P$. The results have a clear tendency: in the majority of runs, additional mutation improves the results. Furthermore, it is shown that the mutation application rate of $m = 0.1$ is preferable.

Using a mutation application rate of $m = 0.1$, the MAs have been run on a variety of problem instances contained in TSPLIB, to show the robustness and scalability of the memetic approach. In Table 7.6, the results are shown for five instances up to a problem size of 1002. The population size was set to $P = 40$ in all runs and the recombination application rate was set to 0.5 and the diversification rate to 0.1. Two MAs were run on each instance, the first one with DPX recombination and the second one with GX recombination. In the latter, *cRate* was set to 1.0, *nRate* was set to 0.1 which appears to be a good compromise between 0.25 and 0.0, and *iRate* was set to 0.5. The programs were terminated as soon as

Table 7.6: Average Running Times of two MAs to find the Optimum

Instance	**Op**	**gen**	**quality**	N_{opt}	**t in s**
lin318	DPX	19	42029.0 (0.000%)	30/30	8
	GX	13	42029.0 (0.000%)	30/30	8
pcb442	DPX	824	50778.0 (0.000%)	30/30	147
	GX	286	50778.0 (0.000%)	30/30	68
att532	DPX	560	27686.0 (0.000%)	30/30	127
	GX	289	27686.0 (0.000%)	30/30	106
rat783	DPX	122	8806.0 (0.000%)	30/30	26
	GX	136	8806.0 (0.000%)	30/30	35
pr1002	DPX	333	259045.0 (0.000%)	30/30	112
	GX	182	259045.0 (0.000%)	30/30	98

they reached an optimum solution. In the table, the average number of generations (gen) and the average running time of the algorithms (t in s) in seconds is provided. In 30 out of 30 runs, the optimum could be found for all instances in less than two minutes. The average running time for rat783 is much lower than for att532 which is not surprising since the landscape of the random instance rat783 has a higher FDC coefficient. The MA with greedy recombination appears to be slightly superior to the MA with DPX in most cases. For larger instances, the average time to reach the optimum as well as the deviation of the running time increases dramatically. Thus, the MA were run on the larger instances with a predefined time limit. Table 7.7 summarizes the results for the MA with greedy recombination (GX). The population size was set to $P = 100$ for pr2392 and pcb3038 since smaller population size

Table 7.7: Performance of MA-GX on large TSP instances

Instance	**gen**	**quality**	**sdev.**	N_{opt}	**t in s**
pr2392	2407	378032.6 (0.000%)	0.8	27/30	2588
pcb3038	5248	137702.6 (0.006%)	6.4	3/30	6955
fl3795	341	28794.7 (0.079%)	21.3	1/30	7212

led to worse performance. Due to long running time of the LK heuristic, the population size for fl3795 was set to $P = 40$. In the table, the average number of generations evolved by the MA (gen), the average final tour length, the percentage access over the optimum solution value (in parentheses), the standard deviation of the final tour length (sdev.), the number of times the optimum was found (N_{opt}), and the running time in seconds (t in s) is provided.

The running times presented here cannot be compared directly with results of alternative approaches found in the literature, since different hardware/software platforms have been used. However, it appears that the MA presented here outperforms other approaches. With ASPARAGOS96 [123], an average tour length of 8809 (0.03%) could be found in approx.

3 hours on a SUN UltraSparc with 170 MHz for rat783, and an average final tour length of 28820 (0.34%) for fl 3795 in approx. 17 hours. The results are significantly worse in both running times and solution quality. With the *Edge Assembly Crossover EA* [234], the running time for finding the optimum for rat783 is 3013 seconds on a PC with a 200 MHz Intel Pentium processor which is much slower even taking the performance differences of the processors into account. The running time to reach a solution quality of 0.006 % for pr2392 is 33285 seconds with their approach which is worse than the MA presented here in both quality and time.

The physically inspired IPT approach [225] outperforms the MA on problem fl3795, for which it requires 6050 seconds on a HP K460 Server with 180 MHz PA8000 processors to find the optimum solution. However, the MA is superior on the instances att532, rat783, and pr2392 in terms of average solution quality. For the latter instance, IPT required 9380 seconds to reach an average final tour length of 378158 (0.033 %).

The *Genetic Iterated Local Search* approach (GILS) [165] is similar to the MA presented in this chapter. Due to the different hardware platform and different running times, a comparison is not possible. GILS delivers very impressive results for the instance pr2392: an average quality of 0.006 % – the optimum is found 3 out of 10 times – is achieved in 1635 seconds on a Fujitsu S-4/5 workstation (microSPARCII 110 MHz). The average final quality for att532 and rat783 is 0.056% and 0.022 % found in 113 and 103 seconds, respectively. However, the MA is able to find the optimum for fl3795 while the optimum could not be found in 26958 seconds by the GILS.

All other heuristics proposed for the TSP, such as simulated annealing [303, 153], tabu search [90], ant colonies [112, 78, 287], artificial neural networks [253, 223, 81], search space smoothing [130], and perturbation [55] have been applied only to rather small problems from TSPLIB or to randomly generated problems. None of these heuristics has been applied to TSPLIB instances between 3000 and 4000 cities.

The *Branch & Cut* approach by Applegate *et al.* [11, 12] required 80829 seconds for pcb3038, and 69886 seconds for fl3795 on a Compaq XP1000 (500 MHz) machine, which is more than two times faster than a Pentium III 500 MHz.

Large Instances

Finally, the MA has been applied to the largest instances in TSPLIB. For these instances, there are no published results of heuristic methods known to the author. Table 7.8 shows the tour length of the optimum solutions as well as the computation time required by *Branch & Cut* to find the optimum [12] on a Compaq XP1000 (500 MHz) machine. The estimated time on a Pentium II 600 MHz PC is provided in the last column to allow a comparison with the running times provided in this chapter. For the three largest problems, the optimum solutions are not known. Therefore, the bounds in which the optimum is known to lie is provided instead of the optimum value itself.

To demonstrate the applicability of the algorithms to very large instances, the MA has been applied to the seven problems listed in Table 7.8. With the same parameters as above, but with termination before the third restart, the MAs were run with a population size P of 10, 20, and 40. The results are presented in Table 7.9. For each population size (P) and each instance, the average number of generations (gen), the average final tour length and percentage access over the optimum or the lower bound (quality), the standard deviation (sdev), and the average time (t) in seconds of 10 runs is displayed.

Table 7.8: The largest instances in TSPLIB

Instance	Optimum/Bounds	Time to find the optimum	
		XP1000 500 MHz	Pentium II 600 MHz
fnl4461	182566	53420.13 sec	≈ 108044 sec
pla7397	23260728	428996.2 sec	≈ 867661 sec
rl11849	923288	≈ 155 days	≈ 313 days
usa13509	19982859	≈ 4 years	≈ 8 years
d18512	$[645198, 645255]$	– open –	
pla33810	$[66005185, 66059941]$	– open –	
pla85900	$[142307500, 142409553]$	– open –	

Table 7.9: Performance of MA-GX on the largest instances in TSPLIB

P	**Instance**	**gen**	**quality**	**sdev.**	**t in s**
10	fnl4461	291	183762.7 (0.655%)	192.1	105
	pla7397	887	23328499.5 (0.291%)	21931.7	802
	rl11849	314	931333.5 (0.871%)	1417.2	417
	usa13509	466	20186311.8 (1.018%)	17135.1	790
	d18512	379	653474.3 (1.283%)	381.3	930
	pla33810	1386	66575838.8 (0.864%)	57687.2	3443
	pla85900	2216	143596390.7 (0.906%)	103234.6	12314
20	fnl4461	528	183366.3 (0.438%)	163.7	294
	pla7397	1155	23307621.7 (0.202%)	14120.4	1860
	rl11849	536	928115.5 (0.523%)	795.8	1006
	usa13509	1082	20125182.2 (0.712%)	27980.9	2422
	d18512	1226	650803.2 (0.869%)	477.8	2873
	pla33810	3832	66321344.7 (0.479%)	45162.4	11523
	pla85900	9069	142986675.5 (0.477%)	79510.3	52180
40	fnl4461	856	183047.1 (0.263%)	82.2	742
	pla7397	1185	23294046.2 (0.143%)	12538.2	3789
	rl11849	861	926253.7 (0.321%)	605.5	2503
	usa13509	1936	20057767.0 (0.375%)	10176.8	6638
	d18512	2091	649354.6 (0.644%)	501.6	7451

The results show that a running time smaller than an hour is sufficient to reach a quality of less than 1% for all problems except the largest one. For the latter, the running time increases to 12000 seconds. Increasing the population size increases the final solution quality, but running times increase drastically. In the extreme case – the largest problem, the running times grow 4.2 times from 12314 to 52180 seconds. In most other cases the running time grows less than 3 times. It can be observed that the pla-problems are better solved than the other instances with respect to the solution quality.

To allow a comparison to a non-population-based approach, ILK has been applied to the seven problem instances. The running times were chosen to allow a comparison with the MA at $P = 20$. Table 7.10 shows the results of the experiments. For each instance, the average

Table 7.10: Performance of ILK on the largest instances in TSPLIB

Instance	iter	quality	sdev.	t in s
fnl4461	7108	183191.1 (0.343%)	72.7	300
pla7397	1830	23324376.2 (0.273%)	17985.5	1800
rl11849	11274	926139.9 (0.309%)	772.9	1000
usa13509	9912	20063763.7 (0.405%)	13400.8	2400
d18512	22243	647949.3 (0.426%)	229.1	2900
pla33810	7930	66270531.2 (0.402%)	22368.1	7200
pla85900	19437	142919653.4 (0.430%)	54291.6	14400

number of iterations (iter), the average final tour length and percentage access over the optimum or the lower bound (quality), the standard deviation (sdev), and the average time in seconds (t in s) of 10 runs is displayed. Except for pla7397, ILK produces better results than the MA. The reason may lie in the fact that in ILK, more local minima are visited. In the MA with $P = 20$, 12 local searches per generation are performed, while in ILK the number of local searches is given by the number of iterations. Thus, the MA performs 13860 local searches compared to the 1830 in ILK for problem pla7397. For instance d18512, ILK visits 22243 local optima, while the MA visits only 14712 in the same time. Generally, the MA appears to be effective on very large instances only if a long running time is spent. The much simpler ILK appears to be the better choice if good solutions are required in short time. However, the results indicate that if high quality solutions ($< 0.2\%$) are required, extremely long running times of several hours to a few days have to be spent for both methods. Taking into account that calculating the optimum for usa13509 took more than 8 years, the running times in the MA experiments are considered fairly small.

The long running times favor the parallel execution of the heuristics in workstation clusters, since compared to the computation times the communication times are small. Here, MAs appear to be better suited than ILK, since they have more potential for concurrent execution. However, besides distributed MAs, distributed ILK algorithms are issues for future research.

7.5 Summary

In this chapter, the fitness landscape of various (Euclidean) traveling salesman problem (TSP) instances has been investigated. The autocorrelation analysis as described in chapter 4 is well suited to find the most effective family of local search algorithms but it does not allow to predict the performance of meta-heuristics based on local search. Therefore, a fitness distance correlation analysis of local optima has been conducted on various TSP landscapes. It has been shown that there are different types of landscapes although the majority of instances has common characteristics: locally optimum tours have more than three third edges in common. Thus, the local minima are contained in a small fraction of the search space. Fractal instances are artificial; they have a highly correlated landscapes and are thus easily solved by simple heuristics. Although they are of interest in the worst case analysis of heuristics [229], they are not well suited for testing highly effective heuristic approaches for the TSP. Random instances in which the the cities are uniformly distributed have got higher correlated local optima with respect to *fast 2-opt* and *Lin-Kernighan local* search than others based on real city coordinates. The local optima of instances in which the cities form clusters – as found in the application of drilling holes in printed circuit boards – have even lower correlation of tour length and distance to the global optimum. These instances belong to the hardest type of instances from the viewpoint of heuristics for the TSP.

The high correlation of tour length and distance to the optimum of the local optima in the TSP landscape is an indicator for a good performance of recombination-based search algorithms, since recombination is capable of exploiting this correlation in an efficient way. However, for the TSP, an effective combination of local search and mutation exists – iterated local search. In an extensive study, several recombination operators including a newly proposed generic greedy recombination operator (GX), are compared against each other in a memetic algorithm framework. The MAs show significant performance differences if a simple *fast 2-opt* local search is employed. For MAs with the sophisticated Lin-Kernighan local search, the results lie much closer together. The study has shown that respectfulness is the most important property of a recombination operator. Furthermore, the MA with the newly proposed greedy recombination operator has been shown to outperform all its competitors: MAs with DPX or MPX recombination, MAs with non-sequential fur change mutation, and iterated local search.

MAs with DPX and GX recombination and mutation have been applied to various instances contained in TSPLIB to show robustness and scalability of the approach. While for problems with up to 1000 cities the optimum could be found in all runs in an average time of less than two minutes on a state of the art personal computer, for the larger instances much more time was required to find the optimum solution. However, for a problem size up to 3795, the optimum could be found in less than two hours. Compared to other proposed approaches, the memetic algorithm appears to be superior in average solution quality and running times. Finally, the MA with GX has been applied to very large instances up to 85900 cities and is thus the first meta-heuristic known to the author which can tackle very large problems.

Chapter 8

The Graph Bipartitioning Problem

8.1 Introduction

The graph bipartitioning problem (GBP) is a combinatorial problem in which a graph has to be partitioned into two equal sized sets by minimizing the number of edges connecting vertices in the different sets.

In this chapter, memetic algorithms for the GBP are discussed. First, two types of heuristics are described that are well suited to be incorporated into a memetic algorithm: greedy heuristics and *k-opt* local search heuristics. A landscape analysis is performed afterwards, employing the techniques described in chapter 4. For solutions generated with the differential greedy heuristic as well as for solutions obtained by the Kernighan-Lin local search, a fitness distance correlation analysis is conducted. The analysis reveals that the types of graphs investigated have extremely different characteristics, ranging from totally uncorrelated landscapes to highly correlated search spaces. For random geometric graphs, it is shown that the correlation of cut size and distance to the optimum is higher for solutions generated by the differential greedy heuristic than for locally optimum solutions with respect to Kernighan-Lin local search if the average vertex degree of the graph is low. For random geometric graphs with higher density (average vertex degree), Kernighan-Lin local optima exhibit a higher correlation of cut size and distance to the optimum. This is surprising, since graph density can be regarded as an indicator for epistasis in the problem. It is shown that with increasing epistasis, the number of local minima decreases in case of random geometric graphs, in contrast to *NK*-landscapes for which it was shown that with increasing epistasis the number of local optima increases as discussed in chapter 5. Completely random graphs, however, do not show such a property. Furthermore, it is shown that for regular graphs there are completely different landscapes in terms of the distribution of local optima. For some graphs, cut size and distance to the optimum are highly correlated while for others, there is no correlation at all.

In experiments it is shown that depending on the type and size of a graph either the combination of differential greedy and Kernighan-Lin local search is sufficient to find (near) optimum partitions or a memetic algorithm is required. The MA is shown to be one of the best approaches developed for the GBP up till now.

The results of this chapter have been published in [217].

8.2 Heuristics for the GBP

Several exact solution approaches to graph partitioning have been proposed, but due to the $\mathcal{NP}$-hardness, their practical usefulness is limited to fairly small problem instances. For example, a branch & cut algorithm proposed in [42] has been applied to instances with no more than 100 vertices. To find partitions of larger graphs, several heuristics have been developed that are capable of producing (near) optimum solutions in reasonable time. Among these are (a) heuristics especially developed for graph partitioning, such as the Kernighan-Lin algorithm [171], greedy algorithms [21], inertial algorithms [271], spectral partitioning [135], multilevel approaches [163], and (b) general-purpose heuristic optimization approaches, such as simulated annealing [157], tabu search [22] and genetic algorithms [47, 284, 141, 149, 184]. Greedy and local search heuristics are described in more detail, since they are used in the fitness distance correlation analysis subsequently, and in the MAs for the GBP.

In the next paragraphs, it is assumed that the cost function to be minimized is given as

$$c(V_1, V_2) = |e(V_1, V_2)|, \text{ with } \quad e(V_1, V_2) = \{(i, j) \in E : i \in V_1 \wedge j \in V_2\}, \tag{8.1}$$

where $c(V_1, V_2)$ is referred to as the cut size of the partition and $e(V_1, V_2) \in E$ is referred to as the (edge) cut.

8.2.1 Greedy Heuristics

The *differential greedy heuristic (Diff-Greedy)* [21] has been shown to be a good constructive heuristic which is capable of producing partitions with a small cut size in short time. Furthermore, Diff-Greedy has shown to be superior to other greedy algorithms, including *Min-Max Greedy* [22] and the standard greedy algorithm with random extraction [22, 21].

Diff-Greedy randomly builds partitions from scratch by adding alternately vertices to one of the sets until each vertex is included in either the first or the second set. The selection which vertex is added to a set in each step is aimed to minimize the difference between new edges across the cut and new internal edges: a vertex j with $D(j, set) = \min_i D(i, set)$ is chosen randomly among the candidates assuming $D(\cdot)$ is defined as:

$$\begin{aligned} D(i, set) &= |E(i, \overline{set})| - |E(i, set)|, \text{ with} \\ E(i, set) &= \{(i, j) \in E \mid j \in set\}, \end{aligned} \tag{8.2}$$

where $set \in \{1, 2\}$, and $\overline{set} = 1$ if $set = 2$, and 2 otherwise. Thus, for a vertex that is included, the difference between the number of adjacent vertices in the other set, and the number of vertices in the same set is minimal. This explains the name differential greedy.

8.2.2 Local Search

The simplest form of local search can be realized by repeatedly exchanging one vertex from set 1 with one vertex from set 2, and accepting the resulting solution if it has a lower cut size than the current one. Thus, in the simplest case, the *1-opt* neighborhood of a partition s is defined as the set of partitions that can be obtained by exchanging one vertex from set 1 with one vertex from set 2 in the solution s.

The Kernighan-Lin Heuristic

Analogously to the TSP, this local search can be extended by exchanging $k > 1$ vertices from set 1 with k vertices from set 2. Kernighan and Lin [171] have proposed a heuristic (*KL heuristic*) for the GBP that exchanges a subset of set 1 with a subset of set 2. Since the size of a *k-opt* neighborhood grows exponentially with k, a small size of this neighborhood is searched utilizing a gain criterion similar to the LK heuristic for the TSP. Thus, the size of the subsets exchanged in each step varies. Fiduccia and Mattheyses [89] proposed a data structure to store the gains achieved when moving a vertex to the other set to speed up the search for a move with the highest gain. With this data structure, the KL algorithm runs in $\Theta(|E|)$ instead of $O(|n^2|)$ per iteration.

The Kernighan-Lin heuristic is one of the best state-of-the-art algorithms for the GBP [157], and it is used in other algorithms for graph partitioning such as multi-level approaches [163].

Bit-Flip Local Search

Alternatively, a local search can be realized by moving just one vertex from one set to another to reach a neighboring solution, allowing the partitions to be unbalanced. Since solutions may thus become infeasible, the objective function has to be redefined:

$$c'(V_1, V_2) = |e(V_1, V_2)| + \alpha(|V_1| - |V_2|)^2, \tag{8.3}$$

where α is called the imbalance factor [157]. With this cost function, imbalanced partitions are penalized depending on the distance to a feasible solution and the imbalance factor. The advantage of this approach is that the neighborhood size of $\mathcal{N}'_{1\text{-}opt}$ reduces to n: two points in the search space are neighbors ($d' = 1$) if they differ in a single bit. On the other hand, the search space is enlarged. The difficulty with penalty functions is to find suitable parameters – in this case the imbalance factor – so that the search can become effective.

For the GBP, a suitable value for the imbalance factor can be derived mathematically as shown below.

8.2.3 Hybrid Evolutionary Algorithms

Several hybrid evolutionary algorithms have been developed for the GBP. For example, Bui and Moon [46, 47] propose an evolutionary algorithm with local search. They use a five–point crossover and bit-flip mutation as evolutionary variation operators and a straightforward binary representation. Since their operators do not guarantee feasible offspring, a random repair scheme is embedded. Their local search is the Kernighan–Lin algorithm, but it is limited to perform only one pass (iteration). Thus, the members of the population are not truly local minima with respect to the KL heuristic. Furthermore, they perform a breadth first search to reorder the vertices on the chromosome before starting the genetic algorithm.

Steenbeek et al. [284] have proposed a hybrid genetic algorithm based on a cluster emplacement heuristic (CEH). In a preprocessing phase, the clusters of the input graph are identified, and each solution is encoded as a set of these clusters. A value of 0 (1) of the i-th gene indicates that cluster i is contained in set 0 (1). The used genetic operators are uniform crossover and bit–flip mutation, both not guaranteed to produce balanced partitions. The cluster emplacement heuristic acts as a combination of local search and repair by iteratively exchanging clusters between the two sets.

Inayoshi and Manderick [149] have proposed a genetic local search algorithm using a variant of uniform crossover for the weighted graph bipartitioning problem. In the weighted GBP, to each edge a real value, called weight, is assigned. The objective is to minimize the total weight of the edges contained in the cut. The encoding is a straightforward binary representation and the crossover is a variant of the uniform crossover, which is guaranteed to produce balanced partitions. The authors use the Kernighan-Lin heuristic to perform local search on each member of the population.

8.3 The Fitness Landscape of the GBP

A solution to the GBP can be encoded in a binary bit string x of length $n = |V|$, where a 0 at locus i indicates that vertex v_i belongs to set 1 and a 1 at locus j indicates that vertex v_j belongs to set 2. The cut size of the partition defined by V_1 and V_2 thus becomes

$$c(V_1, V_2) = c(x) = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{1}{2}w_{ij}|x_i - x_j| \tag{8.4}$$

with $V = \{1, \ldots, n\}$ and $w_{ij} = 1$ if $(i, j) \in E$, 0 otherwise. Because the GBP is a minimization problem, the objective function $c(x)$ has to be minimized, and it is referred to $c(x)$ as the cost function rather than the fitness function. However, the GBP can be turned into a maximization problem by defining the fitness as $f(x) = c_{max} - c(x)$.

8.3.1 A Distance Measure

With the representation described above, the distance metric d can be defined as the hamming distance [131] between two bit strings. If operators are used that guarantee the feasibility of the bit strings, another distance measure appears to be more useful. The smallest possible change in the genotype is to flip two genes simultaneously, a 0 to a 1 and a 1 to a 0. Hence, a vertex from one set is exchanged with one vertex from the other set and the constraint $|V_1| = |V_2|$ is obeyed. Exchanging more than one vertex from set 1 with vertices from set 2 leads to a greater genotypic distance, thus the distance between two solutions x and y can be defined as the minimum number of vertices to exchange between set 1 and set 2 to transform one solution into the other. Considering the hamming distance d_H between bit strings, we have

$$d(x, y) = \min\left(\frac{d_H}{2}, \frac{n - d_H}{2}\right), \tag{8.5}$$

with $d_{min} = 1$, and $d_{max} = \frac{n}{4}$. In contrast to other distance metrics [149, 280], this distance function takes into account that the cut is identical for the two partitions $x = \{1, 1, 1, 1, 0, 0, 0, 0\}$ and $y = \{0, 0, 0, 0, 1, 1, 1, 1\}$.

Neighborhoods for the local search algorithms described above can be easily defined based on this distance metric. The *k-opt* neighborhood is defined as

$$\mathcal{N}_{k\text{-}opt}(s) = \{s' \in S : d(s, s') \le k\}. \tag{8.6}$$

The size of the neighborhood $\mathcal{N}_{1\text{-}opt}$ is $\frac{n^2}{4}$ and in the general case $|\mathcal{N}_{k\text{-}opt}| = \sum_{i=1}^{k}\binom{n/2}{i}^2$. The size of the search space is $|S| = \frac{1}{2}\binom{n}{n/2}$.

8.3.2 Autocorrelation Analysis

The correlation functions of the GBP landscape $\mathcal{L} = (S, c, d)$ induced by the distance metric $d(x, y) = \frac{d_H}{2}$ has been determined mathematically in [280] for randomly generated graphs:

$$\rho(d) = 16 \left(\frac{d}{n}\right)^2 \left(1 + \frac{1}{n-2}\right), \quad r(s) = \left(1 - \frac{8}{n} + \frac{8}{n^2}\right)^s. \tag{8.7}$$

Thus, the correlation length of the landscape becomes [280]:

$$\ell = \frac{1}{8}(n - 3) + O(\frac{1}{n}). \tag{8.8}$$

Surprisingly, the autocorrelation function and the correlation length are independent of the average vertex degree of the graph.

In [181], Krakhofer and Stadler show that for the GBP the expected number of local optima in a ball with radius $R(\ell)$ is 1, with $R(s) = \frac{n}{4}\left[1 - \left(1 - \frac{4}{n}\right)^s\right]$ denoting the expected distance reached after s steps of a simple random walk. They have proven experimentally that the probability that a solution is a local optimum is approximately μ^n with $\mu = 0.609058...$

An Alternative Landscape

An alternative landscape to the landscape $\mathcal{L}$ defined above is the landscape $\mathcal{L}' = (S', c', d')$ which is based on the neighborhood $\mathcal{N}'_{1\text{-}opt}$ described above. The advantage of considering this landscape is that the neighborhood size of $\mathcal{N}'_{1\text{-}opt}$ reduces to $n - 1$: two points in the search space are neighbors ($d' = 1$) if they differ in a single bit. A disadvantage is that the optimum value for the imbalance factor α is not known in advance. However, the optimum value for α for random graphs has been obtained mathematically by Angel and Zissimopoulos [9]: the autocorrelation coefficient λ is maximal if α is set to $p/4$, with p denoting the edge probability $p = \frac{2|E|}{|V|(|V|-1)}$ of the given graph. Furthermore, the authors have shown that the autocorrelation coefficient and hence the correlation length of the alternative landscape $\mathcal{L}'$ is higher than the correlation length of the landscape $\mathcal{L}$.

8.3.3 Fitness Distance Correlation Analysis

Since in general, the fitness distance correlation can not be calculated mathematically, experiments have to be conducted to estimate the FDC coefficient ϱ and to produce the fitness distance plots. To perform the fitness distance analysis for the GBP, several instances with different characteristics have been selected.

The first set of test instances studied in the experiments is taken from Johnson et al. [157], since these instances have been used by several researchers to test their algorithms and thus are a good basis for comparing the results presented in this chapter. Two types of randomly generated instances were considered in [157]:

The first type (a) is denoted $\mathsf{G}n.p$, where n represents the number of nodes and p the probability that any given pair of vertices in the graph constitutes an edge; the expected average degree is thus $p(n - 1)$. The second type (b) is a random geometric graph denoted $\mathsf{U}n.d$, where n is the number of nodes and d the expected average vertex degree. The coordinates of the vertices are chosen randomly from within the unit square and only vertex pairs with a squared distance smaller or equal to $\frac{d}{n\pi}$ constitute an edge to the graph.

The second set of benchmark graphs has been provided by Bui and Moon [47] and consists of (c) regular graphs denoted Breg.n.b with n vertices and optimum cut size b [45], (d) caterpillar graphs denoted Cat.n and Rcat.n with a known optimum cut size of 1 [46], and (e) regular grid graphs (grid.n.b and W-grid.n.b) with optimum bisection size b.

The last set of graphs (f) consists of 2– and 3–dimensional meshes also studied in [134].

The number of edges emanating from the vertices determines the amount of epistasis in a given problem instance. This becomes obvious when the objective function $c(x) = c(V_1, V_2)$ is rewritten as a sum of fitness contributions of each site of the genome $x \in \{0,1\}^n$ representing the partition (V_1, V_2):

$$c(V_1, V_2) = c(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{1}{2} w_{ij} |x_i - x_j| = \sum_{i=1}^{n} c_i(x_i, x_{i_1}, \ldots, x_{i_{k(i)}}) \tag{8.9}$$

with $V = \{1, \ldots, n\}$ and $w_{ij} = 1$ if $(i,j) \in E$, 0 otherwise. Similar to the *NK*-landscapes defined in [169], the fitness contribution c_i of a site i depends on the gene value x_i and of $k(i)$ other genes $x_{i_1}, \ldots, x_{i_{k(i)}}$, where $k(i)$ denotes the degree of vertex i. While for *NK*-landscapes $k(i) = K$ is constant for all i, $k(i)$ varies in the GBP instances introduced above.

Fitness Distance Correlation of Kernighan-Lin Local Optima

In the first experiment, the theoretically determined correlation length for the landscapes of all types of graph was verified. In all cases, the experimentally estimated correlation length coincides with the theoretical value, as given in equation (8.8). Since this value is independent of the average vertex degree or the structure of the graph, it is ill–suited for predicting the performance of evolutionary algorithms on the different types of graph.

To assess the relation between fitness and average distance to the optimum (or optima), the average distance of 2500 points in the search space to all known optima against the cut size difference $\Delta c = c(x) - c_{opt}$ is plotted.

In a first FDA experiment, the points in the search space were generated by the Kernighan-Lin heuristic [171] on random starting solutions. The Kernighan-Lin heuristic is used here since it has been proven to be among the best heuristics for graph partitioning and it is used later in the memetic algorithm for the GBP. Since there are no solutions for the instances of type (a), (b), and (f) proven to be optimum solutions, the experiments were performed using the best-known solutions for the graphs, which are likely to be the optimum solutions.

Additional characteristics of the landscapes are summarized in Table 8.1. Along with the number of vertices $|V|$ and the average vertex degree δ of the analyzed graphs, the number of (global) optima N_{opt} found by the algorithms and the average distance between these optima (in braces) is provided. Furthermore, the average distance to the found optima $\langle d_{opt} \rangle$, the average cut size difference $\langle \Delta c \rangle$, the number of different local optima N_{KL}, the fitness distance correlation coefficient ϱ and the average distance between the local minima $\langle d_{other} \rangle$ are given.

The scatter plots of the landscapes of the graphs of type (a) look quite different compared to the ones of the graphs of type (b), as can be seen in Figure 8.1, although both are randomly generated.

For the G1000 and U1000 graphs, the average cut size increases with higher average vertex degree. This is not surprising, since the fitness contribution c_i of each vertex increases with higher epistasis. But the average cut size difference to the optimum cut size does not increase continuously for the U1000 graphs. Furthermore, the number of distinct local optima

Table 8.1: Average distances and fitness distance coefficients for KL local minima

Instance	$\lvert V\rvert$	δ	$N_{opt}(\langle d\rangle)$	$\langle d_{opt}\rangle$	$\langle\Delta c\rangle$	N_{KL}	ϱ	$\langle d_{other}\rangle$
G1000.0025	1000	2.5	66 (8.5)	212.9	33.4	10000	0.37	225.3
G1000.005	1000	5.0	3 (1.0)	219.7	56.5	10000	0.22	227.9
G1000.01	1000	10.1	175 (62.2)	215.4	73.6	10000	0.37	226.0
G1000.02	1000	20.2	1 (0)	212.4	103.1	10000	0.47	224.7
U1000.05	1000	4.8	2032 (28.1)	216.2	74.4	10000	0.28	224.7
U1000.10	1000	9.4	10 (2.7)	191.6	123.5	9999	0.36	206.5
U1000.20	1000	18.7	1 (0)	129.4	99.3	4542	0.63	157.6
U1000.40	1000	36.0	1 (0)	94.5	131.1	624	0.82	130.2
hammond.graph	4720	5.8	1 (0)	632.7	51.5	9385	0.54	754.1
Breg5000.16	5000	3.0	1 (0)	104.2	163.0	9117	0.99	197.7
Cat.5252	5252	2.0	1 (0)	1255.5	252.6	10000	0.02	1267.8
RCat.5114	5114	2.0	1 (0)	1143.6	183.8	10000	0.07	1165.7
Grid5000.50	5000	3.9	1 (0)	119.8	10.3	2201	0.91	214.8
W-grid5000.100	5000	4.0	12 (618.9)	677.8	24.4	6519	0.66	721.5

decreases, as well as their average distance to each other: the higher the epistasis, the closer the local minima in the search space. The same effects can not be observed for the G1000 instances: the average distance to the optima is relatively constant and much higher than for the random geometric graphs. The number of optima found for the U1000 graphs decreases with the average vertex degree, and more than 2000 optima could be found for U1000.05. These optima are located in a small region of the search space since their distance to each other is small. For three G1000 instances, more than one optimum solution could be found, but the highest number could surprisingly be found for G1000.01.

The mesh hammond of graph type (f) exhibits a similar structure as the random geometric graphs with average vertex degree 10 and 20, as shown in Figure 8.2. The random regular graph of type (c) shows a high correlation of distance and cut size. Most of the local optima lie close to the optimum, as the average distance to the optimum and the average cut size difference suggest. The caterpillar graphs (d) are again completely different. The graph Cat.5252 does not show a correlation between cut size and distance to optimum and the Kernighan-Lin local optima lie on the average with maximum distance apart from the optimum. The scatter plot for the instance RCat.5114 has no similarity to the plot for Cat.5252. The nearest local minimum has a distance greater than 600 to the optimum. The last type of instances (e), the grid graphs, also have an interesting distribution of local optima. The local optima of regular grid graphs seem to have high correlation (ϱ = 0.91) between cut size and distance to optimum, while for the wrapped-around grid graphs the correlation is significantly lower. A possible reason for this is that the optima for the wrapped–around grid have a relatively high distance (618).

The random geometric graphs have a structure that may be exploited by a search algorithm, as mentioned in [157]. While the adjacent vertices of a given vertex are completely random for the graphs of type (a), the adjacent vertices have a small geographical distance, for all other graphs.

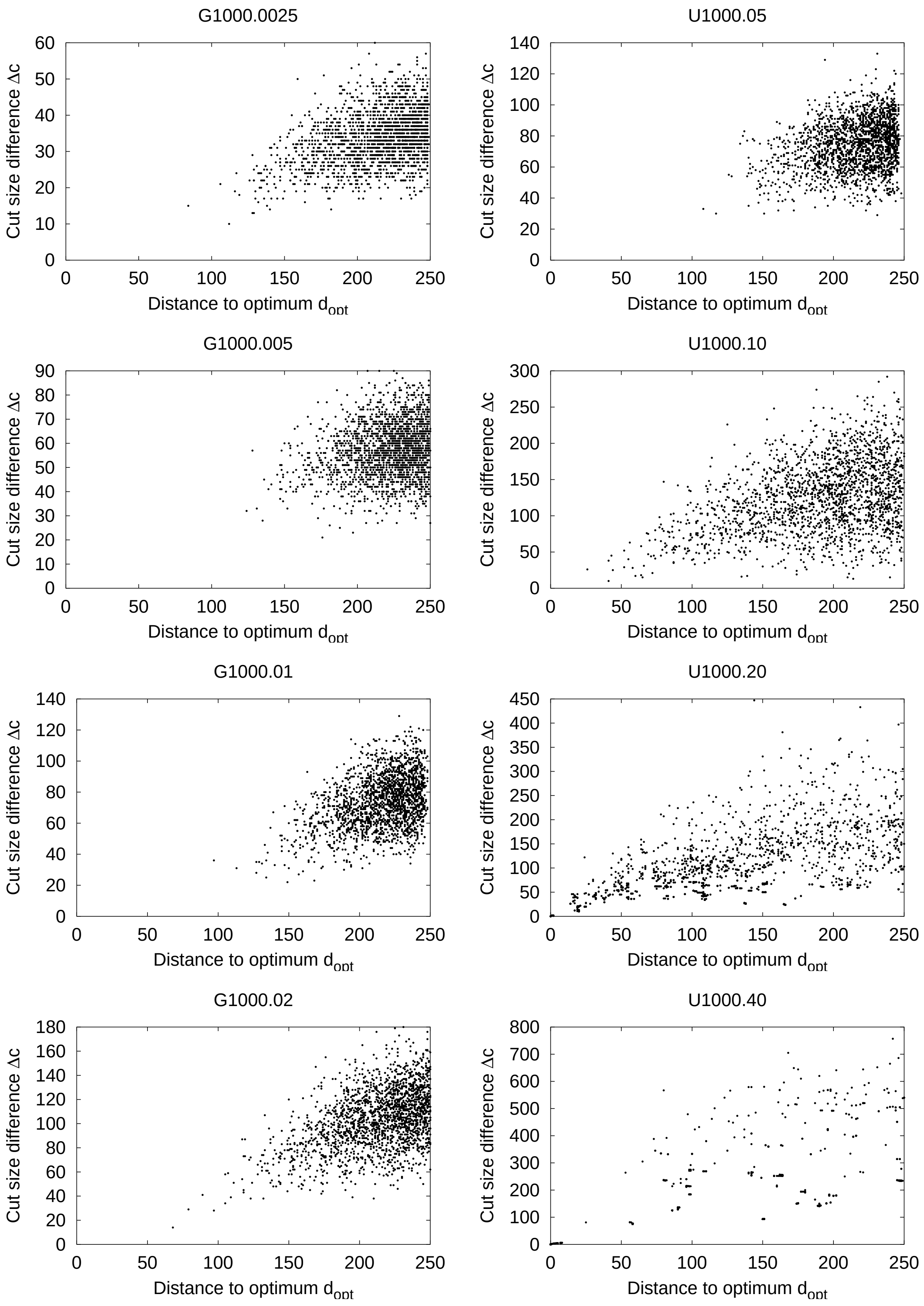


Figure 8.1: G1000.d (left) and U1000.p (right) fitness-distance plots with KL

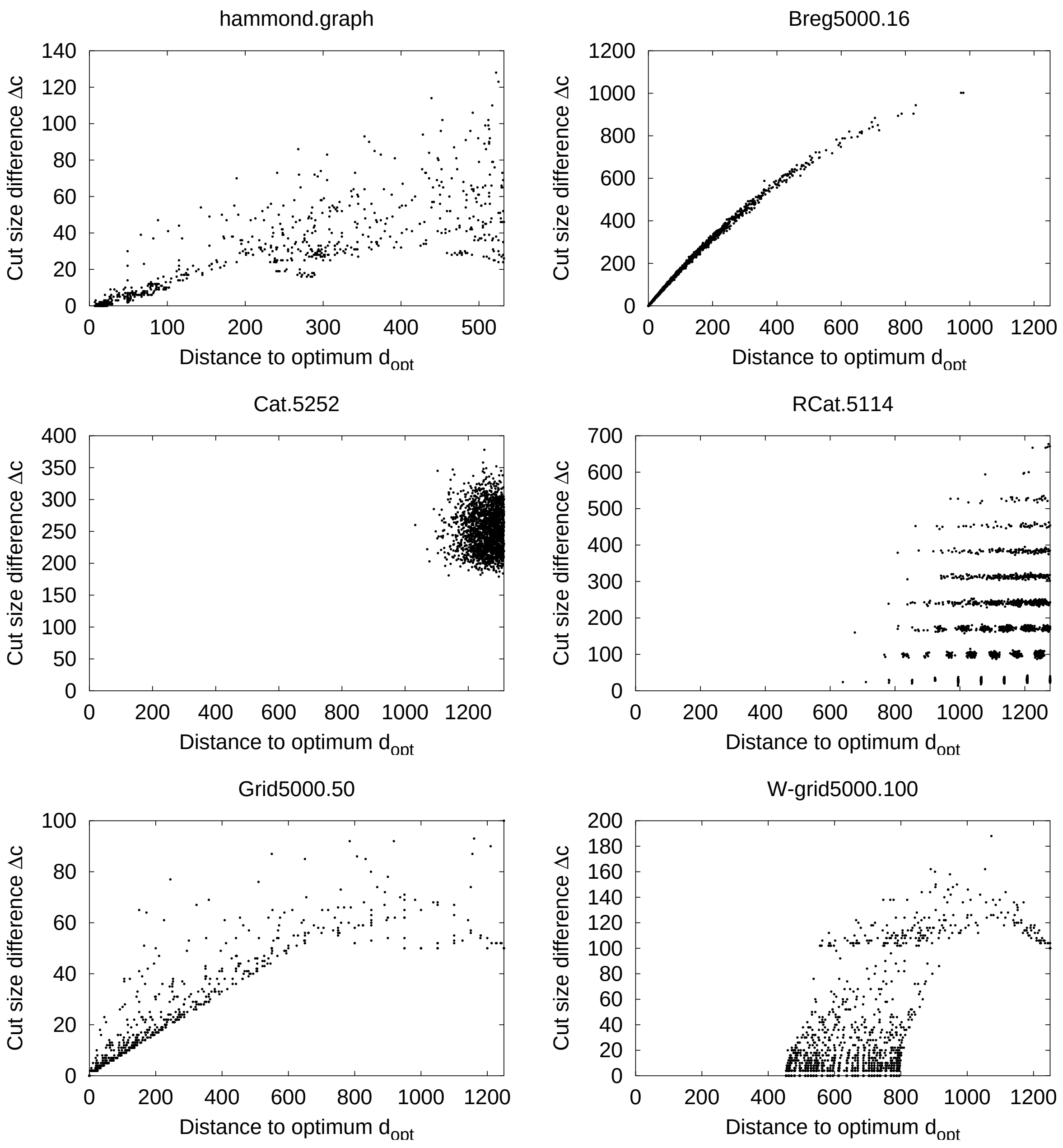


Figure 8.2: Fitness-distance plots using KL for graphs of type (c) to (f)

Fitness Distance Correlation of Diff-Greedy Solutions

In the second FDA, the points in the search space were generated with the differential greedy algorithm [21]. The scatter plots are provided in Figure 8.3, and in Table 8.2 the average distance to the found optima $\langle d_{opt} \rangle$, the average cut size difference $\langle \Delta c \rangle$, the number of different Diff-Greedy solutions N_{DG}, the fitness distance correlation coefficient ϱ and the average distance between the solutions $\langle d_{other} \rangle$ are given.

The Diff-Greedy algorithm performs poorly on the type (a) graphs but performs well on the type (b) graphs. It performs better than the Kernighan-Lin algorithm on the random geometric graphs, but the latter is superior on uniform random graphs. The Diff-Greedy

Table 8.2: Average distances and fitness distance coefficients for Diff-Greedy solutions

Instance	$\langle d_{opt} \rangle$	$\langle \Delta c \rangle$	N_{DG}	ϱ	$\langle d_{other} \rangle$
U1000.05	111.5	10.1	9998	0.63	144.4
U1000.10	128.1	45.8	9415	0.58	153.9
U1000.20	135.3	143.4	7981	0.58	149.4
U1000.40	104.7	265.3	5174	0.66	136.0
G1000.0025	208.6	43.0	10000	0.34	220.9
G1000.005	216.8	79.2	10000	0.18	223.6
G1000.01	213.6	118.0	10000	0.29	222.1
G1000.02	208.6	177.0	10000	0.41	222.0
hammond.graph	674.1	106.7	10000	0.33	744.5
Breg5000.16	599.1	455.0	9994	0.99	895.1
Cat.5252	605.1	3.3	8221	0.21	765.6
RCat.5114	585.8	6.4	6709	0.70	739.4
Grid5000.50	604.3	76.8	9391	0.70	753.3
W-grid5000.100	677.6	32.6	8594	0.70	721.6

algorithm is capable of exploiting the structure of the graphs (c) - (f), too. Optima or near–optimum solutions are found easily by Diff–Greedy, even for the caterpillar graphs – even though the distribution of Diff–Greedy solutions for these instances shows no correlation.

Discussion

The analysis has shown that the landscape of the GBP is highly dependent on the structure of the graph. The analysis of a single graph is not sufficient to predict the performance of an algorithm on other graphs, hence the insights gained by the analysis performed in [149] for the weighted graph bipartitioning problem are probably rather limited. It seems likely that certain distributions of weights may also have a strong influence on the properties of the fitness landscape of the weighted graph bipartitioning problem.

The average vertex degree, or in other words the amount of epistasis in the problem is only one characteristic of a graph influencing the structure of the fitness landscape. Other aspects such as regularity (graphs of type (c) to (e)) and the locality of epistatic interactions are also important.

Gene interaction in a given representation can appropriately be expressed by a dependency graph. An edge in the dependency graph from vertex i to vertex j indicates that the fitness contribution f_i of gene i depends on the value of x_j. Thus, the fitness contribution of gene i is of the form $f_i(x_i, \ldots, x_j, \ldots)$. For *NK*-landscapes defined by Kauffman, the vertex degree of the dependency graph is $K + 1$, including the edges going from the vertices to themselves. It appears that the structure of the dependency graph may have a large impact on the fitness landscape. In the GBP, the dependency graph is similar to the graph to be partitioned, with the difference that the latter does not contain edges from a vertex to itself. The fitness landscape analysis of the GBP has shown that the structure of the dependency graph is responsible for the structure of the fitness landscape. If the dependency graph contains only edges between neighboring points and thus induces a spatial structure, the

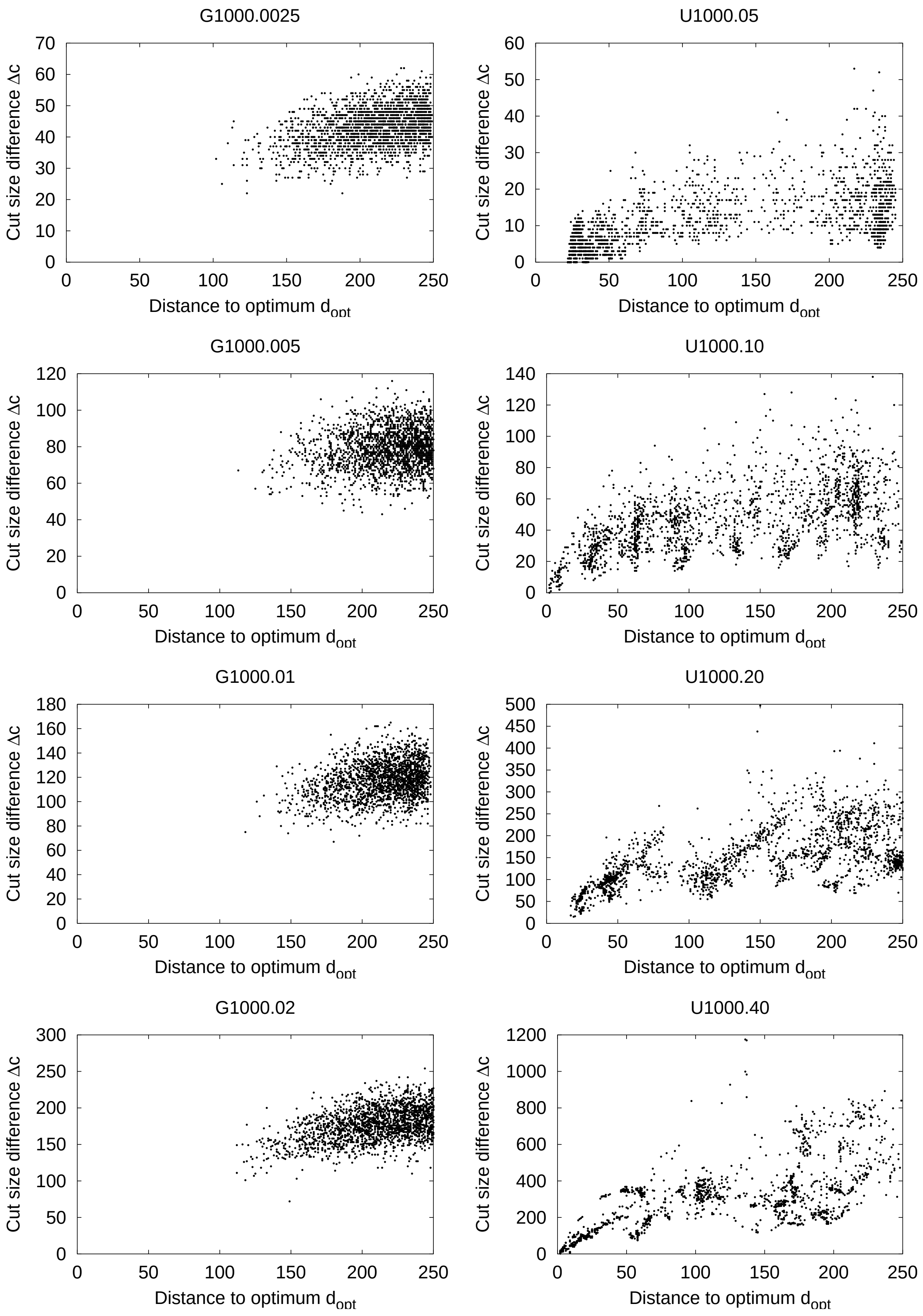


Figure 8.3: G1000.d (left) and U1000.p (right) fitness-distance plots with Diff–Greedy

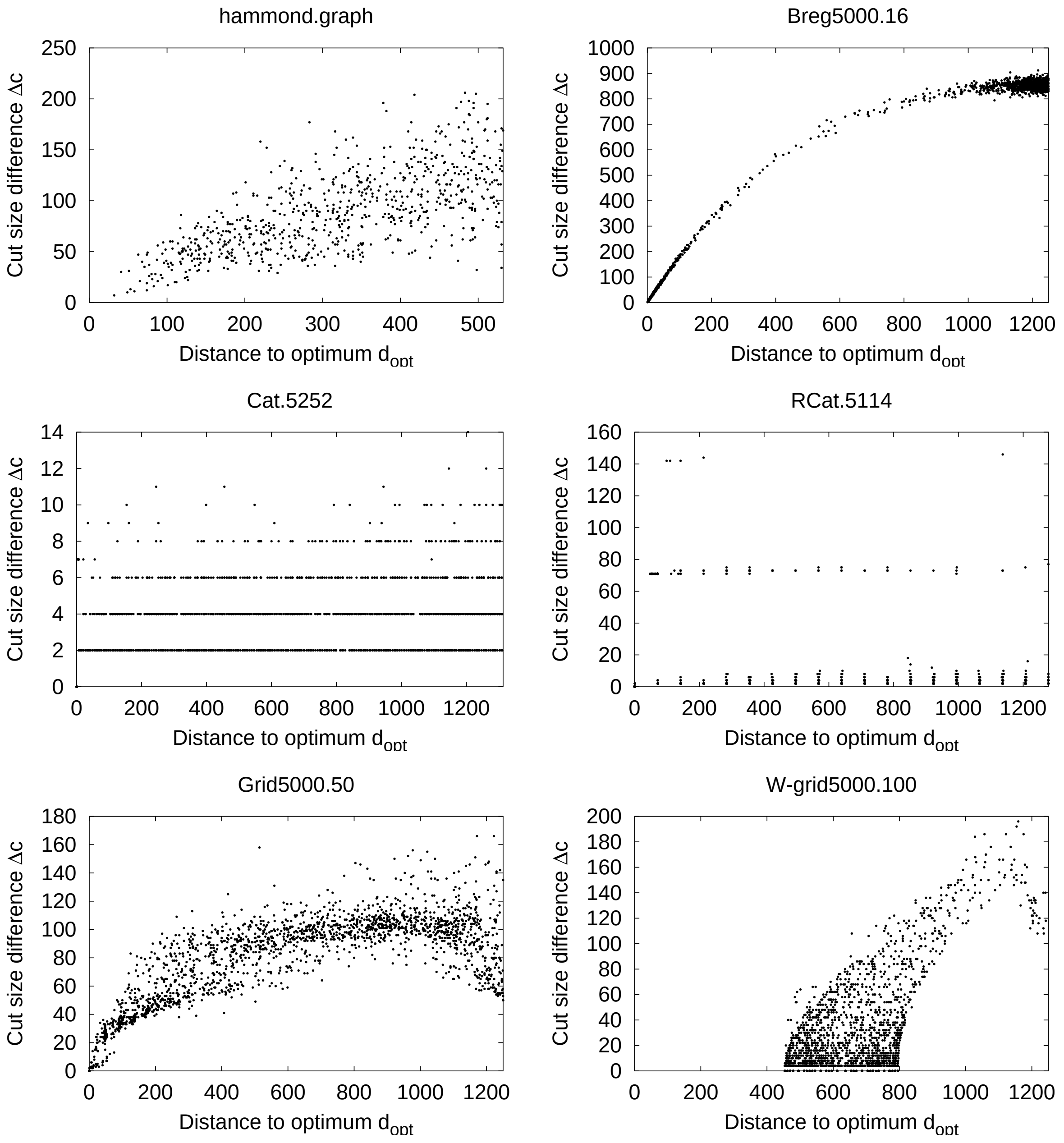


Figure 8.4: Fitness-distance plots using Diff–Greedy for non–random graphs

resulting landscape is quite different from a landscape resulting from a dependency graph with edges randomly sampled over all vertex pairs, as can be seen from the FDC plot for the U1000 and G1000 graphs, respectively. If there is regularity in the dependency graph, the fitness landscape may show a high correlation between fitness and distance to an optimum (see the plot for instance Breg5000.16). Other forms of regularity lead to a totally uncorrelated landscape (Cat.5252).

8.4 A Memetic Algorithm for the GBP

A memetic algorithm employing the greedy and local search heuristics is described in the following. Since the outline of the memetic algorithm for the GBP is identical to the general description in chapter 3, the description focuses on the problem–specific parts: the initialization of the population, the local search, and the evolutionary variation operators.

8.4.1 Initialization and Local Search

Instead of generating the starting solutions randomly, the randomized differential greedy heuristic may be used, since it is one of the best construction heuristics for the GBP and is capable of producing a wide range of high quality solutions.

Three different local search algorithms for the GBP were considered: The 1-opt local search based on the neighborhood $\mathcal{N}_{1\text{-}opt}$ (SWAP), the alternative neighborhood $\mathcal{N}'_{1\text{-}opt}$ (FLIP) as described above, and the Kernighan-Lin algorithm which efficiently searches a small part of the $\mathcal{N}_{k\text{-}opt}$ neighborhood. In the case of FLIP, it is not guaranteed that the resulting local optimum of a local search is a feasible solution. Hence, a repair algorithm as described in [157] is applied to turn the solution into a feasible one. Table 8.3 shows the typical performance in terms of average cut sizes (of 10000 runs) and time per local search (LS) on random starting solutions of the three local search algorithms for four different graphs. The problem instances, computing platforms, and experimental conditions are described in detail in section 8.5.

Table 8.3: Average cut sizes and CPU times (Pentium II - 300 MHz) per local search

	SWAP		**FLIP**		**KL**	
Graph	cut size	time/LS	cut size	time/LS	cut size	time/LS
G1000.0025	232.0	1.49 ms	186.2	1.96 ms	126.2	16.41 ms
G1000.005	625.6	2.09 ms	583.6	2.62 ms	501.3	21.01 ms
G1000.01	1585.2	3.36 ms	1567.7	3.91 ms	1435.5	37.07 ms
G1000.02	3663.8	6.02 ms	3646.9	6.75 ms	3484.7	72.11 ms

All algorithms utilize a data structure based on the ideas of Fiduccia and Mattheyses [89], to speed up the search for a swap with the highest gain. Instead of using a single gain list, two different gain lists are used for the two sets. Thus, the KL algorithm used in the experiments is exactly the one proposed by Kernighan and Lin with the difference that it runs in $\Theta(|E|)$ instead of $O(|n^2|)$ per iteration.

Not surprisingly, the FLIP algorithm performs better than the SWAP algorithm, which could be expected from the higher correlation length of the FLIP landscape L', as mentioned in the previous section. Since the Kernighan-Lin local search is capable of finding partitions with a significantly smaller cut size, the decision was made to use this algorithm in the memetic approach.

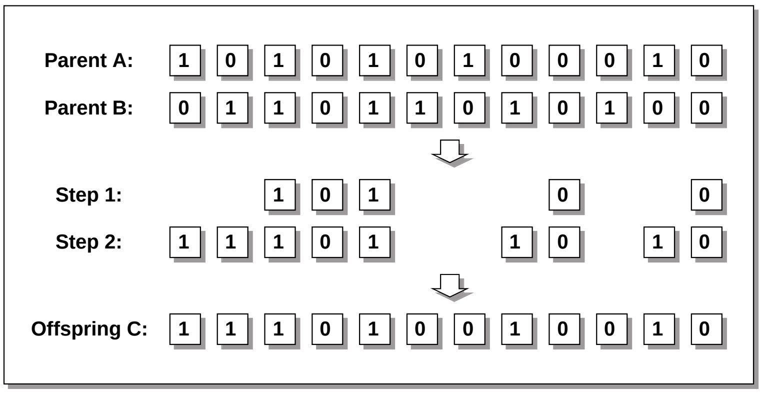
8.4.2 The Evolutionary Variation Operators

Mutation for the GBP can be realized by exchanging a randomly chosen subset of size s of vertices from partition 1 with a randomly chosen subset of vertices from partition 2. The sizes of the subsets determine the distance between a solution and the mutated offspring.

Traditional crossover operators such as one–point, two–point, and uniform crossover cannot be applied without modification to guarantee feasibility. For example, valid crossover points for one or two–point crossover are those where the number of ones on the left side of the crossover point is identical in both parents. Restricting the operator to cut the parents only at these cut points assures the feasibility of the generated offspring.

Uniform crossover can be modified to operate on GBP solutions as follows. First, all 1s that are found at the same locus in both parents are copied to the offspring. Then, additional 1s are inserted into the offspring by randomly selecting a locus from one of the parents containing a one, until the offspring consists of $n/2$ 1s and hence is feasible.

Another crossover operator for the memetic algorithm is the HUX [86], a variant of the uniform crossover. In order to maintain feasibility, HUX must also be adapted for application to the GBP. HUX can be performed by first copying all 0s and 1s that are found at the same location in both parents. The remaining entries are filled with 1s in random order from alternating parents until $\frac{n}{2}$ 1s are included. Those entries that are still not set will be filled with 0s. An example illustrating the functionality of HUX is given below:


Parent A: 1 0 1 0 1 0 1 0 0 0 1 0
Parent B: 0 1 1 0 1 1 0 1 0 1 0 0
Step 1: 1 0 1 0 0
Step 2: 1 1 1 0 1 1 0 1 0
Offspring C: 1 1 1 0 1 0 0 1 0 0 1 0

Thus, HUX is biased so that nearly as many 1s are taken from the first as from the second parent.

A fourth recombination operator was developed called *greedy recombination operator (GX)* based on the differential greedy algorithm. The pseudo code for the GX operator is provided in Figure 8.5.

In the first phase, all vertices that are contained in the same set in both parents are included in the same set in the offspring. Then, both sets are filled according to the selection scheme in the Diff–greedy algorithm. If $|V_1| < |V_2|$, a vertex is added to V_1, otherwise to V_2.

8.5 Memetic Algorithm Performance

Several experiments were conducted to test the memetic algorithm on 48 instances, 8 graphs of type (a) and (b), 12 graphs of type (c), 8 graphs of type (d) and (e), and 4 graphs of type (f).

The memetic algorithm described above, including the Kernighan-Lin heuristic and the differential greedy heuristic, has been implemented in $C{+}{+}$. All experiments were performed

```
procedure GXrecombination($a \in S$; $b \in S$): $S$;
  begin
    for $i := 1$ to $n$ do
      if $a[i] = b[i]$ then $c[i] := a[i]$ else $c[i] := -1$;
    endfor
    $n0 :=$ number of 0s in child $c$;
    $n1 :=$ number of 1s in child $c$;
    while $n0 + n1 < n$ do
      if $n0 \geq n1$ then
        randomly select $j$ with $D(j,1) = \min_i D(i,1)$;
        $c[j] := 1$;
        $n1 := n1 + 1$;
      else
        randomly select $j$ with $D(j,0) = \min_i D(i,0)$;
        $c[j] := 0$;
        $n0 := n0 + 1$;
      endif
    endwhile
    return c;
  end;
```

Figure 8.5: The GX recombination operator

on a Dual Pentium II PC (300 MHz) with Solaris 2.6 as operating system (only one processor was used by the algorithms).

In the first set of experiments to test the performance of the memetic algorithm, the MA was applied with 10 different genetic operators to the uniform random instances G$n.p$, since the landscapes of these instances are the most challenging of all landscapes considered in this paper. The first MA, denoted MA-UX is a memetic algorithm with uniform crossover. Two further MAs use one-point and two-point crossover operators and are denoted by MA-1pt and MA-2pt, respectively. The HUX operator is used in the memetic algorithm called MA-HUX. The MA with the greedy recombination operator is called MA-GX; in a variant denoted by MA-RGX, the subsequent LS after recombination is restricted to exchange only those vertices that do not belong to the same set in both parents. In all these algorithms no additional mutation is performed. The algorithms with mutation instead of recombination are denoted MA-M. The size of the subsets exchanged by the mutation operator is varied from $s = 50$ to $s = 200$. Furthermore, runs with the multi–start Diff-Greedy+Kernighan-Lin local search (DG+KL) algorithm and an iterated Kernighan-Lin algorithm (IKL) were performed to put the results into perspective.

The MAs were run with a population size of 40 and a recombination/mutation application rate of 0.5. These parameters have been used in the memetic algorithms for the TSP and the QAP [210, 209]. They are a good a priori choice, but depending on the landscape and the running time of the algorithms, other choices may produce better results. Finding the best parameters for the MA is itself an optimization problem [107], and it is unlikely that there is an optimum parameter set which is best for every instance. On the other hand, determining

Table 8.4: MA results for G500.p instances

algorithm	gen	$\bar{c}$	σ_c^2	N_{opt}	t/s	gen	$\bar{c}$	σ_c^2	N_{opt}	t/s
	G500.005					**G500.02**				
MA-UX	974	50.3	0.76	5/30	60	492	627.0	0.76	6/30	60
MA-1pt	1333	50.3	0.95	9/30	60	639	627.0	0.76	6/30	60
MA-2pt	1335	50.5	0.90	7/30	60	670	627.3	1.15	8/30	60
MA-HUX	901	50.0	0.85	10/30	60	475	627.3	0.99	6/30	60
MA-GX	1252	• 49.1	0.37	29/30	60	633	627.5	1.14	6/30	60
MA-RGX	1985	49.1	0.51	28/30	60	1335	627.4	0.97	4/30	60
MA-M,s=50	709	50.9	0.40	1/30	60	346	• 626.7	0.71	13/30	60
MA-M,s=100	504	51.5	0.51	0/30	60	234	627.0	0.76	8/30	60
MA-M,s=150	489	51.5	0.51	0/30	60	229	627.0	0.76	9/30	60
MA-M,s=200	626	50.9	0.35	0/30	60	317	626.8	0.70	9/30	60
DG+KL	14603	52.0	0.26	0/30	60	5673	627.8	1.45	8/30	60
IKL	26292	55.8	2.11	0/30	60	12925	638.8	4.26	0/30	60
	G500.01					**G500.04**				
MA-UX	676	• 218.0	0.00	30/30	60	226	1745.5	1.50	14/30	60
MA-1pt	909	218.1	0.55	29/30	60	320	1745.9	1.76	11/30	60
MA-2pt	955	218.1	0.37	29/30	60	341	1746.5	2.52	10/30	60
MA-HUX	691	218.2	0.65	28/30	60	223	1745.8	1.65	12/30	60
MA-GX	959	218.1	0.51	28/30	60	326	1745.4	1.50	15/30	60
MA-RGX	1731	218.1	0.51	28/30	60	807	1746.5	2.47	12/30	60
MA-M,s=50	535	218.1	0.37	29/30	60	194	• 1745.3	1.54	12/30	60
MA-M,s=100	372	218.5	0.51	16/30	60	119	1745.6	1.28	7/30	60
MA-M,s=150	363	218.3	0.48	20/30	60	117	1745.3	1.12	7/30	60
MA-M,s=200	476	218.0	0.00	30/30	60	182	1745.8	2.07	11/30	60
DG+KL	9251	219.3	0.84	5/30	60	2815	1747.1	2.12	3/30	60
IKL	19172	229.7	5.21	0/30	60	6984	1763.8	8.67	0/30	60

the best parameter choices for each graph separately will soon become computationally prohibitive. Furthermore, since the influence of the choice of the genetic operators is much higher on the performance of the algorithm, the population size and operator rates were kept constant. Each run was terminated after a predefined time limit. The results are displayed in Tables 8.4 and 8.5. For each algorithm, the average number of generations in case of a MA or the average number of iterations in case of IKL and DG+KL, the average cut size $\bar{c}$, its standard deviation σ_c^2, the number N_{opt} of times an optimum (best-known solution) was found and the time limit t in seconds is provided. The best average cut size for each instance is marked with a dot. All average values are based on 30 runs. In almost all cases, the MA with greedy recombination (GX) performs better than any other crossover–based MA. However, for a high average vertex degree, the mutation based MAs outperform the MAs with crossover. In [212], we have shown that for *NK*-Landscapes with high epistasis, the mutation based MA is more effective than the recombination based MA. The same obviously holds for the GBP. Multi-Start LS (Diff-Greedy + Kernighan-Lin) and IKL are clearly outperformed by the MAs. It appears that the concept of a population–based search is important in case of the GBP, since the IKL based on a single solution does not produce competitive results.

A major drawback of a mutation–based MA is that the optimum mutation operator is not known in advance. To find the optimum distance of a jump performed by mutation, additional experiments were conducted on the G1000 graphs. The results are summarized in

Table 8.5: MA results for G1000.p instances

algorithm	gen	$\bar{c}$	σ_c^2	N_{opt}	t/s	gen	$\bar{c}$	σ_c^2	N_{opt}	t/s
	G1000.0025					**G1000.01**				
MA-UX	802	96.3	0.95	0/30	120	360	1363.4	1.33	8/30	120
MA-1pt	1187	96.9	1.50	0/30	120	519	1366.7	2.95	1/30	120
MA-2pt	1294	96.9	1.53	0/30	120	558	1367.0	3.79	2/30	120
MA-HUX	757	96.1	0.97	0/30	120	360	1363.2	1.48	13/30	120
MA-GX	1031	94.5	1.33	10/30	120	514	• 1363.1	1.04	9/30	120
MA-RGX	1616	• 94.2	1.32	14/30	120	1145	1366.2	3.64	4/30	120
MA-M,s=50	742	96.7	1.18	0/30	120	341	1370.9	3.66	0/30	120
MA-M,s=100	612	96.3	1.03	0/30	120	261	1367.5	3.49	3/30	120
MA-M,s=150	466	97.0	0.76	0/30	120	200	1364.6	2.75	7/30	120
MA-M,s=200	369	99.9	1.03	0/30	120	160	1372.0	2.33	0/30	120
DG+KL	10706	101.4	1.45	0/30	120	3945	1378.1	2.62	0/30	120
IKL	9790	99.5	2.87	0/30	120	4230	1370.8	4.66	2/30	120
	G1000.005					**G1000.02**				
MA-UX	578	449.4	1.99	2/30	120	188	3384.5	1.38	0/30	120
MA-1pt	853	451.6	3.22	1/30	120	291	3385.5	2.40	0/30	120
MA-2pt	936	451.5	3.33	0/30	120	299	3385.3	2.20	1/30	120
MA-HUX	576	449.0	2.24	3/30	120	187	3383.9	0.78	2/30	120
MA-GX	807	• 447.7	0.99	2/30	120	282	3384.0	0.49	0/30	120
MA-RGX	1514	448.1	1.73	4/30	120	742	3385.0	1.27	0/30	120
MA-M,s=50	557	452.2	2.72	1/30	120	201	3389.4	7.00	4/30	120
MA-M,s=100	443	450.4	1.79	0/30	120	147	3386.3	4.51	1/30	120
MA-M,s=150	350	448.9	1.48	0/30	120	110	• 3383.2	0.81	6/30	120
MA-M,s=200	290	457.3	2.20	0/30	120	80	3387.9	2.03	0/30	120
DG+KL	7087	459.9	2.23	0/30	120	2029	3397.5	5.17	0/30	120
IKL	7330	452.9	4.09	0/30	120	2276	3399.5	14.73	1/30	120

Figure 8.6. The optimum mutation jump distance seems to increase with the average vertex degree of the graphs, and for all graphs this distance is near the correlation length of the landscape ($\ell \sim 125$).

In an additional experiment, the best recombination operator (GX) and the best mutation operator ($s = 50$ for G500 and $s = 150$ for G1000) shown in Table 8.4 and 8.5 were combined into a further memetic algorithm, called MA-GXM. The results are shown in Table 8.6, where the first column displays the average cut size and computation time (in seconds) after 30 generations, and the second column shows the average cut size (of 30 runs) after a predefined time limit of 60 and 120 seconds, respectively. Compared to other algorithms, including simulated annealing (SA) [157], the hybrid genetic algorithms CE-GA [284] and BFS-GBA [47], as well as reactive–randomized tabu search (RRTS) [22], the MA with both GX recombination and mutation seems to be superior or at least competitive, as shown in Table 8.6 (the values for simulated annealing are not displayed here, since they are worse than any of the results presented in the table; they can be found in [157, 47, 284]). In particular, the MA outperforms the algorithms SA, CE-GA and BFS-GA on all random non-geometric graphs, and it is better than RRTS on G$n.p$ graphs with low average vertex degree and slightly worse on the graphs with high vertex degree. However, allowing longer running times, the MA is capable of finding smaller cuts than RRTS for the graphs with high average vertex degree, too, as displayed in the second column. The CPU times cannot be compared directly since CE-GA was run on a SGI-O2 workstation (R5000, 180 MHz),

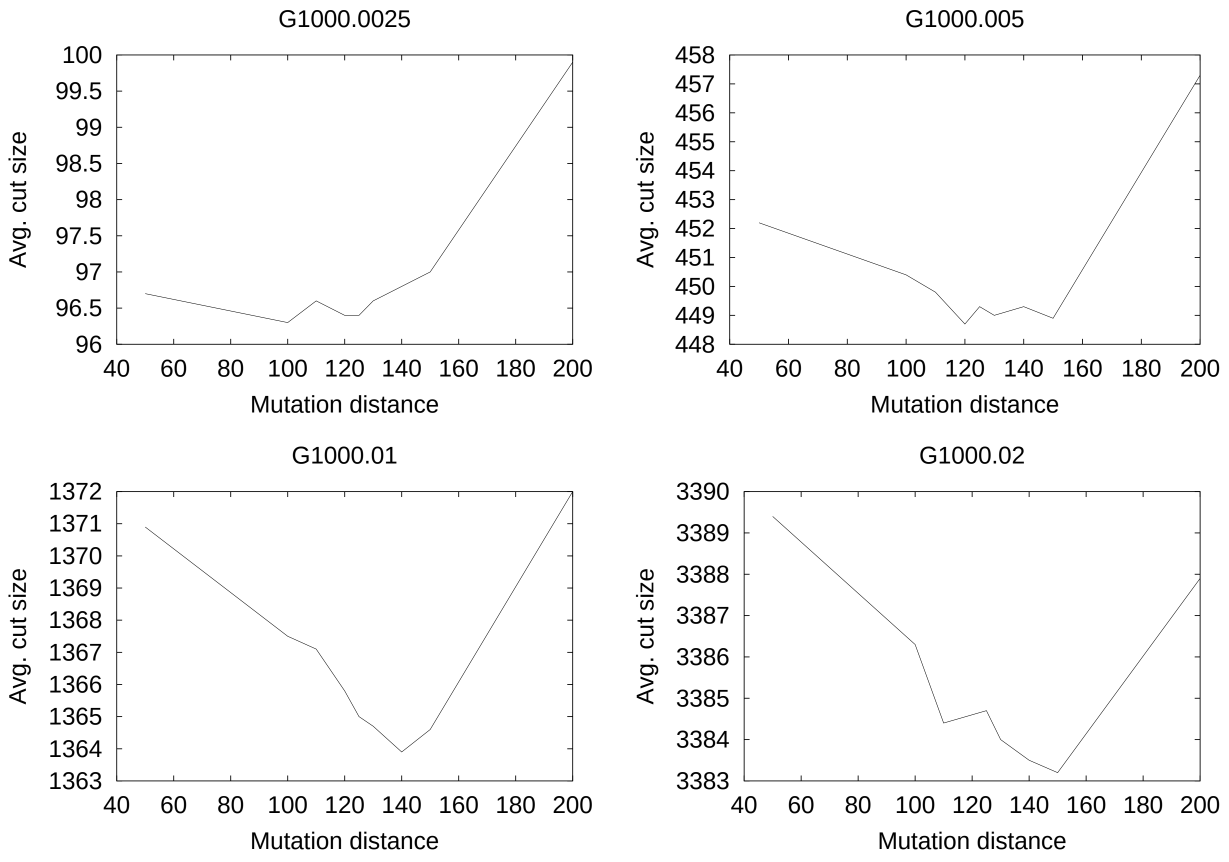


Figure 8.6: Mutation jump distances for the G1000 problems

BFS-GA on a Sparcstation IPX, and RRTS was run on a Digital AlphaServer 2100 Model 5/250 (the time for input/output and the initialization of data structures was not included for the latter).

Table 8.6: Comparison of five algorithms on non-geometric graphs

Instance	**MA-GXM**		**MA-GXM**		**CE-GA**		**BFS-GBA**		**RRTS**	
	cut	time	cut	time	cut	time	cut	time	cut	time
G500.005	50.4	2	49.3	60	54.1	24.9	54.0	6	52.06	2
G500.01	218.2	3	218.1	60	221.8	23.7	222.1	8.1	218.29	2.5
G500.02	627.8	5	626.7	60	631.1	27.5	631.5	11.7	626.44	3.6
G500.04	1746.9	10	1744.9	60	1750.3	33.4	1752.5	21.6	1744.36	6.8
G1000.0025	96.5	7	95.1	120	104.5	79.2	103.6	16.8	98.69	6.5
G1000.005	450.7	10	447.6	120	458.5	79.9	458.6	23.7	450.99	6.5
G1000.01	1365.0	17	1363.3	120	1374.6	79.5	1376.4	37.1	1364.27	9.3
G1000.02	3384.6	31	3383.5	120	3396.8	85.8	3401.7	62.3	3383.92	14.7

The second set of experiments was performed on the geometric graphs of types (b) through (f). Since the Kernighan-Lin algorithm performs better on the random geometric instances of high epistasis and the Diff-Greedy algorithm has advantages on the graphs

Table 8.7: Comparison of four algorithms on geometric graphs

Instance	**DG+KL**		**CE-GA**		**BFS-GBA**		**RRTS**	
	cut	time	cut	time	cut	time	cut	time
U500.05	2.0	0.09	2.2	13.4	3.7	7.5	2.0	1.7
U500.10	26.0	0.55	26.0	10.5	32.7	9.6	26.0	2.7
U500.20	178.0	0.46	178.0	26.3	197.6	11.5	178.0	5.3
U500.40	412.0	0.08	412.0	9.2	412.2	9.9	412.0	10.2
U1000.05	1.0	0.14	3.2	43.3	1.8	17.6	1.0	4.2
U1000.10	39.0	1.59	39.0	20.1	55.8	30.9	39.03	6.3
U1000.20	222.0	0.68	225.9	37.1	231.6	33.0	222.0	12.5
U1000.40	737.0	0.40	738.2	38.1	738.1	37.0	737.0	24.3

with low epistasis as revealed by the fitness distance analysis, the combination of both appears to be a good idea. Thus, the multi–start Diff-Greedy + Kernighan-Lin algorithm (DG+KL) was run on all geometric instances. Instead of limiting the time, the algorithms were stopped as soon as they reached the optimum cut size. Table 8.8 displays the average time (in seconds) to reach the optimum as well as the average number of iterations (iter). For the graphs of type (b) a comparison with other heuristics is provided in Table 8.7. The combination DG+KL is superior to all other algorithms, since it finds the best-known cut sizes in all the runs. Again, the CPU times should be treated with care due to the different hardware/software platforms used.

All running times are less than one second (with one exception), hence the use of a memetic algorithm is not justified. It seems that with increasing problem size, the multi–start approach becomes ineffective and that a MA is superior. In order to check this hypothesis, experiments were conducted summarized in Table 8.9 on large 2D and 3D meshes.

For graph hammond, the multi–start approach is slightly faster, but for graph barth5 ($|V| = 15606$) and brack2 ($|V| = 62631$) the MA proves to be considerably faster. For the largest mesh (ocean) with 143437 vertices, the DG+KL algorithm was not able to find a bisection of size 464, so runs were performed with a predefined time limit of 1200 seconds. The recombination–based MA was able to find the (probably minimal) cut size of 464 four out of 30 times. Average cut size and standard deviation are much smaller for the MA than for DG+KL, indicating that the MA scales much better with the problem size.

8.6 Summary

In this chapter, a fitness distance analysis (FDA) for several instances of the GBP was performed. It has been shown that the structure of the search space of several types of graphs differs significantly: Locally optimum solutions of some graphs show a high correlation between fitness and distance to the optimum (or best-known solution), while for others no correlation between fitness and distance can be observed. Thus, it is not sufficient to perform a FDA for only one or two instances (or type of instance) and to draw conclusions from the obtained results for all other instances, as in previous studies on graph partitioning [149, 34]. This, of course, may also be true for other combinatorial optimization problems.

The search space analysis has shown that neither the correlation length of the landscape,

Table 8.8: Diff-Greedy + Kernighan-Lin on regular, caterpillar and grid graphs

instance	iter	cut size	average t/s
Breg100.0	1.03	0	0.005
Breg100.4	1.30	4	0.005
Breg100.8	4.23	8	0.007
Breg100.20	7.10	16	0.010
Breg500.0	1.13	0	0.019
Breg500.12	5.37	12	0.035
Breg500.16	9.43	16	0.050
Breg500.20	27.10	20	0.113
Breg5000.0	1.13	0	0.183
Breg5000.4	1.30	4	0.216
Breg5000.8	2.07	8	0.270
Breg5000.16	4.53	16	0.513
Cat.352	5.97	1	0.018
Cat.702	6.60	1	0.037
Cat.1052	6.80	1	0.052
Cat.5252	6.73	1	0.273
RCat.134	2.27	1	0.008
RCat.554	4.20	1	0.024
RCat.994	3.87	1	0.041
RCat.5114	6.63	1	0.390
Grid100.10	1.53	10	0.009
Grid500.21	2.23	21	0.023
Grid1000.20	1.60	20	0.049
Grid5000.50	2.67	50	0.361
W-grid100.20	1.77	20	0.007
W-grid500.42	2.27	42	0.026
W-grid1000.40	2.00	40	0.038
W-grid5000.100	2.57	100	0.254

nor epistasis in terms of the average number of interacting genes per site in the problem can help in predicting the structure of the landscape and hence the performance of heuristic algorithms. Therefore, the notion of a dependency graph was introduced describing the spatial structure of the gene interactions of the underlying representation. It has been shown that the regularity of the dependency graph has a high influence on the shape of the fitness landscape of the graph bipartitioning problem. Furthermore, locality (i.e., the dependency graph contains only edges between vertices near to each other) seems to be an important property. In the GBP, randomly generated geometric instances are shown to have a more structured landscape than purely random instances with a dependency graph exhibiting no spatial structure.

Algorithms such as the differential greedy algorithm exploit the structure of the search space and are thus very effective. A new greedy recombination operator has been developed

Table 8.9: Diff–Greedy vs. MA on large 2D/3D meshes

algorithm	graph	gen/iter	cut size	avg. time/s
DG+KL	hammond.graph	19.67	90	1.523
MA-GX,P=10	hammond.graph	3.73	90	1.580
DG+KL	barth5.graph	1151.77	139	424.962
MA-GX,P=20	barth5.graph	21.60	139	43.943
DG+KL	brack2.graph	145.47	731	473.503
MA-GX,P=40	brack2.graph	4.20	731	255.199

algorithm	graph	gen	cut size	sdev.	N_{opt}	time/s
DG+KL	ocean.graph	260	475.9	5.12	0/30	1200
MA-GX	ocean.graph	28	467.8	1.58	2/30	1200
MA-RGX	ocean.graph	72	467.2	1.81	4/30	1200

which is based on the same idea and can be easily embedded in a memetic algorithm. It has been shown in computer experiments that the MA with greedy recombination is superior to five MAs using other crossover operators, and also superior to multi–start local search and iterated local search on random uniform graphs. However, for graphs with high average vertex degree, the memetic algorithm using mutation instead of recombination proves to be superior. This phenomenon can also be observed for NK-Landscapes as shown in chapter 5 where it also appears to be better to use mutation instead of recombination for high K and hence high epistasis. A MA in which the greedy recombination operator is combined with mutation delivered the best results for uniform random graphs with low *and* high epistasis. This algorithm is superior or at least competitive to the best currently available heuristics the author is aware of.

For geometric graphs, the combination of differential greedy and Kernighan-Lin local search is sufficient for small graphs (up to 5000 nodes). It was shown that for larger graphs the memetic algorithm is much more effective. Thus, the memetic algorithm proves to be an efficient, scalable and very robust search algorithm on all types of graphs and is capable of producing better average cut sizes than every other heuristic search method known to the author, including simulated annealing, hybrid genetic algorithms, and tabu search.

Chapter 9

The Quadratic Assignment Problem

9.1 Introduction

The quadratic assignment problem (QAP) is a combinatorial problem in which a set of facilities with given flows has to be assigned to a set of locations with given distances in such a way that the sum of the product of flows and distances is minimized.

In this chapter, a fitness landscape analysis for several instances of the quadratic assignment problem is performed and the results are used to classify problem instances according to their hardness for memetic algorithms. The local properties of the fitness landscape are studied by performing an autocorrelation analysis, while the global structure is investigated by employing a fitness distance correlation analysis as described in chapter 4. It is shown that epistasis, as expressed by the dominance of the flow and distance matrices of a QAP instance, the landscape ruggedness in terms of the correlation length of a landscape, and the correlation between fitness and distance of local optima in the landscape together are useful for predicting the performance of memetic algorithms to a certain extent. Thus, based on these properties a favorable choice of recombination and/or mutation operators for MAs can be found.

Furthermore, experiments comparing three different evolutionary operators for a memetic algorithm are presented. It is shown in an extensive comparison study that a memetic algorithm employing the recently proposed information-preserving recombination operator for the QAP is able to outperform five of its competitors, two variants of tabu search, two ant colony algorithms, and simulated annealing, on all tested instances of practical interest on a set of problem instances with a problem size of up to 256.

The results presented in this chapter have been published in [216].

9.2 Heuristics for the QAP

Several heuristics have been proposed for finding near–optimum solutions to large QAP instances, including *ant colonies* [199, 111, 290, 288], *evolution strategies* [237], *genetic algorithms* [300], *simulated annealing* [50, 56], *neural networks* [150], *memetic algorithms* [41, 152, 93, 231] *tabu search* [23, 273, 274, 296, 14], *threshold accepting* [236], *randomized greedy search* (*GRASP*) [189], *scatter search* [61], and *tree search* [198].

In the following, the usefulness of greedy construction heuristics for the QAP is discussed, and local search variants for the QAP are described since both types of algorithms are relevant for the development of effective MAs for the QAP.

Throughout this chapter, it is assumed that a solution to the QAP is written as a permutation π of the set $\{1, \ldots, n\}$. The cost associated with a permutation π is

$$C(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}\, b_{\pi(i)\pi(j)}, \tag{9.1}$$

where n denotes the number of facilities/locations, and $A = (a_{ij}), B = (b_{ij})$ are referred to as the distance and flow matrices, respectively.

9.2.1 Greedy Heuristics

Greedy construction heuristics have been proposed for several combinatorial optimization problems. However, for the QAP these algorithms are fairly limited.

A construction heuristic for the QAP works by assigning facilities to locations until all facilities are assigned. Thus, a greedy construction heuristic requires N steps to produce a feasible QAP solution for an N facility/location problem. The crucial part of such a heuristic is the criterion that determines which facility to assign to which location in a construction step. The criterion is aimed to maximize a ‘profit’ associated with each possible assignment and is only meaningful if the profit is observed in the resulting solution. Thus, the profit can be regarded as the contribution to the final objective value of the solution. Usually, QAP instances have a high degree of non-linearity. Except for instances with distance and flow matrices with low density, an assignment of a facility to a particular location influences many other choices and the profit estimated for prior assignments becomes totally different from the real contribution to the final objective. Hence, a greedy heuristic performs in most cases *not* considerably better than random search but requires much more running time and thus cannot be justified as an alternative to random search.

9.2.2 Local Search

The commonly used local search for the QAP is the the *2-opt* heuristic, also known as the pairwise interchange heuristic [43].

In the QAP, the *2-opt* neighborhood is defined as the set of all solutions that can be reached from the current solution by swapping two elements in the permutation π. Figure 9.1 illustrates such a *2-opt* move. The number of swaps and consequently the size of this

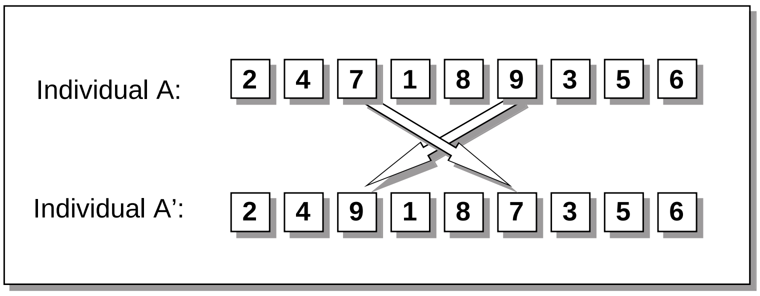


Figure 9.1: Local Search for the QAP

neighborhood grows quadratically with n. The change in the total cost $C(\pi) - C(\pi')$ by a swap of the elements i and j in the permutation π can be calculated in linear time:

$$\begin{aligned}\Delta C(\pi,i,j) &= C(\pi) - C(\pi') \\ &= (a_{jj}-a_{ii})(b_{\pi(i)\pi(i)}-b_{\pi(j)\pi(j)}) + (a_{ji}-a_{ij})(b_{\pi(i)\pi(j)}-b_{\pi(j)\pi(i)}) \\ &\quad + \sum_{k=1,k\neq i,j}^{n} (a_{jk}-a_{ik})(b_{\pi(i)\pi(k)}-b_{\pi(j)\pi(k)}) + (a_{kj}-a_{ki})(b_{\pi(k)\pi(i)}-b_{\pi(k)\pi(j)}).\end{aligned} \tag{9.2}$$

However, the formula is reduced considerably if both matrices A and B are symmetric and all the diagonal elements of one of the matrices are zeros:

$$\Delta C(\pi,i,j) = 2\sum_{k=1,k\neq i,j}^{n} (a_{jk}-a_{ik})(b_{\pi(i)\pi(k)}-b_{\pi(j)\pi(k)}). \tag{9.3}$$

There are several techniques to speed up the neighborhood search. The most commonly used technique in tabu search is to maintain a cost gain matrix ΔC. Instead of calculating each element of the matrix in each iteration anew, the gain matrix can be updated efficiently by utilizing the following formula assuming that the solution π' has been obtained by exchanging facilities k and l in the solution π:

$$\begin{aligned}\Delta C(\pi',i,j) &= \Delta C(\pi,i,j) \\ &\quad +(a_{ki}-a_{kj}+a_{lj}-a_{li})\cdot(b_{\pi(l)\pi(i)}-b_{\pi(l)\pi(j)}+b_{\pi(k)\pi(j)}-b_{\pi(k)\pi(i)}) \\ &\quad +(a_{ik}-a_{jk}+a_{jl}-a_{il})\cdot(b_{\pi(i)\pi(l)}-b_{\pi(j)\pi(l)}+b_{\pi(j)\pi(k)}-b_{\pi(i)\pi(k)}).\end{aligned} \tag{9.4}$$

The formula is valid for all $i,j \notin \{k,l\}$. In the other cases, equation (9.2) has to be used instead. Again, for symmetric instances, the formula is less complicated.

Instead of searching for the best neighboring solution, the first swap found reducing the total cost $C(\pi)$ may be accepted. Furthermore, another mechanism may be included to reduce the running time of the local search. The use of *don't look bits* has been described in chapter 7 and has been proposed by Bentley for the TSP [27]. However, the technique can be used in any other local search algorithm for other combinatorial problems, too. In the QAP, the mechanism works as follows. If the don't look bit for location i is set to one, the facility at location i will not be considered for an improvement swap in the current local search iteration. Initially, all don't look bits are set to zero. The don't look bit for location i will be set to one, if no improving move could be found for the current solution with the facility at location i being one of the facilities to swap. If an improving swap is found in which the facility at location k has to be exchanged with the facility at location l, the don't look bits for location k and l are set to zero.

This technique reduces the running time without a significant loss in quality of the solutions.

9.2.3 Hybrid Evolutionary Algorithms

Some evolutionary approaches incorporating local search have been proposed for the QAP, some of which are based on recombination and can be classified as memetic algorithms. These will be briefly described in the following.

In [231], a parallel genetic algorithm is presented that incorporates 2-opt local search. Recombination is performed by a voting mechanism: Each child has p parents. The number of parents in which a facility is assigned to the same location is counted. If the facility is assigned to a location more often than a predefined threshold, the facility is assigned to that location in the child. All other assignments are made at random. Voting recombination is a highly disruptive recombination with a high degree of implicit mutation.

The SAGA algorithm proposed in [41] incorporates simulated annealing instead of a simple 2-opt local search. Recombination is performed similar to the PMX crossover proposed by Goldberg and Lingle [118] for the TSP. A sequence of assignments between two randomly chosen crossover points is copied from the first parent to the offspring. Additional assignments are made that are found in the second parent while maintaining feasibility. The remaining unassigned facilities are randomly allocated. Hence, the operator also performs implicit mutations.

The memetic algorithm described in [152] uses the crossover operator as in SAGA. But here, a tabu search is used instead of simulated annealing.

The Genetic Hybrids introduced in [93] are genetic algorithms incorporating local search or tabu search. The recombination operator used in their algorithms is borrowed from Tate and Smith [300] and works as follows. Firstly, all facilities assigned to the same location in both parents are copied to this location in the child. Secondly, the unassigned locations are scanned from left to right. For the unassigned locations, a facility is chosen at random from those occupying the location in the parents if they are not yet included in the child. Thirdly, remaining facilities are assigned at random. Thus, implicit mutation occurs in the last step of the recombination scheme.

9.3 The Fitness Landscape of the QAP

The binary matrix representation as defined in equation (2.12) is impractical for the coding of the solutions due to the high storage requirements and the complicated constraint handling. Therefore, the permutation π is usually encoded as a vector of facilities, such that the value j of the i-th component in the vector indicates that facility j is assigned to location i ($\pi(i) = j$). Individual A in Figure 9.1 represents a solution where facility 2 is assigned to location 1, facility 4 is assigned to location 2 and so on.

9.3.1 A Distance Measure

The fitness landscape analysis and the memetic algorithm described below rely on a distance metric for QAP solutions. There are several possibilities for measuring distances between permutations. The commonly used distance measure is as follows. Let π_1 and π_2 be valid permutations and hence valid solutions to a given QAP instance. The distance between the solutions π_1 and π_2 is defined as:

$$d(\pi_1, \pi_2) = |\{i \in \{1, \ldots, n\} \mid \pi_1(i) \neq \pi_2(i)\}| \,. \tag{9.5}$$

Thus, the minimum distance between distinct solutions is $d_{min} = 2$, and the maximum distance and therefore the diameter of the landscape is $d_{max} = n$ with n denoting the problem size.

Although there may be alternative landscapes for the QAP, the search space analysis is focused on the landscape $\mathcal{L} = (S, f, d)$ with $S = \Pi(n)$ (the set of all permutations of $\{1, \ldots, n\}$), $f = c(\pi)$ as defined in equation(2.11), and d as defined in equation (9.5), since the local search is designed for that landscape. The diameter $diamG_{\mathcal{L}}$ of the landscape is equal to the problem size n.

9.3.2 Types of QAP Instances

The transformation of instances of a given optimization problem into easier solvable problem instances is an interesting issue for designing heuristics. Thus, the goal is to find transformations that help in increasing the efficiency or the effectiveness of an algorithm. For the QAP, the following transformation may be employed for developing promising heuristics.

QAP matrix transformations

Assuming the matrix B of a QAP instance is symmetric, the matrix $A = (a_{ij})$ can be transformed into $A' = (a'_{ij})$ without changing the resulting cost, if

$$a'_{ij} = \lambda_{ij}\,(a_{ij} + a_{ji}) \quad \text{with} \quad \lambda_{ij} = 1 - \lambda_{ji}, \quad \lambda_{ii} = \frac{1}{2}. \tag{9.6}$$

Thus, setting λ_{ij} to 0.5 for all i, j, the transformation of an asymmetric matrix A yields a symmetric matrix A' with

$$C(\pi) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}\, b_{\pi(i)\pi(j)} = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{1}{2}(a_{ij} + a_{ji})\, b_{\pi(i)\pi(j)} = \sum_{i=1}^{n}\sum_{j=1}^{n} a'_{ij}\, b_{\pi(i)\pi(j)}. \tag{9.7}$$

An analogous transformation can be utilized to obtain two symmetric matrices if matrix A is symmetric while B is not. By using this transformation, the computation time for a local search can be reduced significantly for asymmetric instances. In the memetic algorithm described in this chapter, the transformation is performed once upon startup, and thus the computation time for the local search is reduced considerably (up to a factor of 4 in the implementation) due to the much faster evaluation of ΔC.

The transformation provided in equation (9.6) does not change the structure of the fitness landscape. Assuming matrix A is asymmetric and B is symmetric, the fitness (cost) difference between neighboring points in the search space is given by:

$$\begin{aligned} \Delta C(\pi, i, j) &= C(\pi) - C(\pi') \\ &= (a_{jj} - a_{ii})(b_{\pi(i)\pi(i)} - b_{\pi(j)\pi(j)}) \\ &\quad + \sum_{k=1, k\neq i,j}^{n} (a_{jk} - a_{ik} + a_{kj} - a_{ki})(b_{\pi(i)\pi(k)} - b_{\pi(j)\pi(k)}). \end{aligned} \tag{9.8}$$

Thus, the sum of a_{jk} and a_{kj} always contribute to the cost change between neighboring solutions. The transformation above does not change the sum of two elements a_{ij} and a_{ji}, thus ΔC does not change for a transformed problem, and hence the structure of the fitness landscape remains.

Flow and Distance Dominance

QAPLIB [49] is a publicly accessible library of instances of the QAP. It contains different types of QAP instances; some of them were drawn from real world applications, while others were generated randomly to assess the performance of heuristics. In order to find a complexity measure for QAP instances, Vollmann and Buffa [304] have introduced the *flow dominance*, which measures to what extent the flow matrix B shows "dominant" flow patterns. The dominance $dom(X)$ for a matrix X is defined as the coefficient of variation multiplied by 100:

$$\begin{aligned} dom(X) &= 100 \cdot \frac{\sigma(X)}{\mu(X)}, \\ \mu(X) &= \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}, \\ \sigma(X) &= \sqrt{\frac{1}{n^2 - 1} \sum_{i=1}^{n} \sum_{j=1}^{n} (x_{ij} - \mu(X))^2}. \end{aligned} \tag{9.9}$$

Thus the flow dominance is denoted $dom(B)$. In case a few entries comprise a large part of the overall flow, the flow dominance is high, and if almost all entries are equally sized, the flow dominance is low. However, the major drawback of this complexity measure is that it does not take the influence of the distance matrix A into account. Analogously, let $dom(A)$ be the distance dominance, then the dominance of a QAP instance can be defined according to [7, 8] as a vector $(\min\{dom(A), dom(B)\}, \max\{dom(A), dom(B)\})$.

QAP instances with randomly generated flows (distances) using a uniform distribution typically have a low flow (distance) dominance, whereas real-life instances and (non-uniformly) randomly generated instances similar to real-life instances have considerably higher dominance values for at least one of the matrices.

Epistasis and the QAP

The amount of gene interaction (epistasis) is known to have a strong influence on the performance of heuristics. If the solutions of a problem can be encoded in a bit string (binary vector), and the fitness can be decomposed into fitness contribution functions for each site, epistasis can be estimated easily. Consider the NK-Landscapes proposed by Kauffman [168]. The fitness f of a genome $x = (x_1, \ldots, x_N) \in \{0,1\}^N$ is defined as follows:

$$f(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x_i, x_{i_1}, \ldots, x_{i_K}). \tag{9.10}$$

Thus, the fitness contribution f_i for each site i depends on the gene value x_i of gene i and on the values of K other genes at the sites $i_1, \ldots, i_K$. Hence, the higher the value of K, the higher the epistasis. It is shown in chapter 8, that there are problems for which the quantity denoting the (average) number of interacting genes does not sufficiently reflect the characteristics of the fitness landscape. To gain insight into the role of gene interactions, the notion of a dependency graph reflecting gene interaction is introduced. The vertices in the dependency graph represent the genes. An edge in the dependency graph from vertex i to vertex j indicates that the fitness contribution f_i of gene i depends on the value of x_j. Thus, the fitness contribution of gene i is of the form $f_i(x_i, \ldots, x_j, \ldots)$. For NK-landscapes, the vertex degree of the dependency graph is $K + 1$, including the edges going from the vertices to themselves.

Since the commonly used representation of the QAP is a permutation of the set $\{1, \ldots, N\}$, it is not obvious what gene interaction means in the QAP. The cost function $c(\pi)$ (equation

(2.11)) can be decomposed in the following way:

$$c(\pi) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}\, b_{\pi(i)\pi(j)} = \sum_{i=1}^{n} c_i(\pi) \tag{9.11}$$

Assuming the matrix B does not contain zeros in the off–diagonal entries, a fitness (cost) contribution function c_i depends on those sites j for which $a_{ij} \neq 0$. In other words, the dependency graph is described by the matrix A, where $a_{ij} \neq 0$ defines the weight of the edge (i, j). If matrix B has more elements in the off–diagonal entries equal to zero, the role of the matrices can be exchanged. Note that exchanging the matrices A and B does not change the fitness landscape of a problem instance.

Thus, the dominance measure defined above is a good indicator for the amount of epistasis in a problem instance. For example, assuming matrix B (A) has no or only few zero entries, $dom(A)$ ($dom(B)$) is low and epistasis is high; if matrix A (B) has many zero entries, the dominance is high and epistasis is low.

The Complexity Catastrophe

In his book [168], Kauffman describes a phenomenon called the *complexity catastrophe*. He observed that in NK landscapes the expected fitness of local optima decreases towards 0.5, the mean fitness of the search space, if both K and N increase (K increases as a constant fraction of N).

Burkard and Fincke [48] have shown for the QAP that the ratio between the objective function values of worst and optimal solutions is arbitrarily close to one with probability tending to one as the size of the problem (n) approaches infinity. They have proven this theorem for instances with independently and identically distributed random variables for the entries of the matrices A and B. Hence, the complexity catastrophe occurs also in the QAP at least for instances with uniformly random generated matrices. A consequence of the theorem is that simple heuristics are sufficient for large problem instances, and for infinitely large n, even random search is appropriate.

9.3.3 Autocorrelation Analysis

To analyze the fitness landscape of the QAP, 12 instances were selected from QAPLIB, including real-world instances, randomly generated instances, and instances with known optimum solutions generated by an algorithm proposed in [188]. Furthermore, two instances of the graph bipartitioning problem and two traveling salesman problem instances transformed into QAP instances were included in the analysis. The chosen instances are listed in Table 9.1. Viewing the table from left to right, the name of the instance, the problem size n, and the dominance of matrix A and B ($dom(A)$ and $dom(B)$, respectively) are provided. The last two columns contain the correlation length in relation to the diameter of the landscape (n/ℓ), and the correlation length ℓ itself. The correlation length ℓ has been estimated experimentally by performing random walks of length 100000000.

The first 9 instances (tai80a - tai256c) are either real world or randomly generated QAP instances, the lipa90a/b QAP instances are randomly generated in a way that optimum solutions can be calculated in polynomial time. The two instances beginning with G124 are transformed graph bipartitioning instances. The transformed TSP instances are kroA124p and kroA100, an asymmetric and a symmetric TSP, respectively.

Table 9.1: Average distances and fitness distance coefficients for local minima

Instance	n	$dom(A)$	$dom(B)$	$\langle d_{opt} \rangle$	$\langle \Delta c \rangle$	q	ϱ	$\langle d_{ls} \rangle$	i_{ls}	n/ℓ	ℓ
tai80a	80	59.2	60.4	78.9	547619.7	4.04	0.02	78.9	139.8	4.04	19.60
tai100a	100	59.3	60.3	97.7	776234.7	3.67	0.06	99.0	183.9	4.05	24.71
sko100a	100	50.8	106.6	97.6	3035.6	2.00	0.08	97.9	342.8	3.66	27.29
wil100	100	50.8	64.5	97.8	2909.4	1.07	0.03	97.6	343.2	3.59	27.82
tho150	150	51.5	147.2	147.9	185767.2	2.28	0.04	147.8	632.8	3.68	40.72
tai80b	80	64.0	323.2	77.3	44117586.0	5.39	0.22	77.7	366.9	3.41	23.48
tai100b	100	80.4	321.3	95.2	53694038.8	4.53	0.62	96.7	490.1	2.86	35.01
tai150b	150	51.8	314.1	148.1	15278698.1	3.06	0.05	148.1	839.4	3.75	39.95
tai256c	256	259.7	217.9	254.6	231150.2	0.52	0.05	255.0	193.5	4.04	63.29
lipa90a	90	18.4	41.9	88.8	2834.3	0.79	0.03	89.0	159.0	4.09	22.02
lipa90b	90	60.0	41.9	88.5	2758634.9	22.09	0.37	88.9	161.8	4.05	22.24
G124-02	124	100.0	711.3	123.0	20.7	79.74	0.00	123.0	42.9	4.00	30.97
G124-16	124	100.0	224.7	122.8	47.9	5.33	0.00	123.0	84.6	4.05	30.58
kro124p	100	995.0	49.6	99.0	25043.0	69.12	0.07	99.0	292.9	4.09	24.47
kroA100	100	995.0	54.8	99.1	20840.0	97.92	0.07	99.0	358.3	4.08	24.52

Focusing on the local structure of the landscapes, the instances can be divided into two classes. Several instances have a highly rugged fitness landscape as reflected by a low correlation length ($n/\ell \sim 4$). The instances sko100a, wil100, tho150 and tai*b have a higher relative correlation length as expressed by a lower n/ℓ. The smoothest landscape has instance tai100b with $n/\ell = 2.86$. Some of the rugged landscapes have another interesting property: For the instances tai*a and tai256c, the average number of improvements made by the 2-opt local search is very low compared to the other instances.

The Random Walk Correlation Function

In an additional experiment, the random walk correlation function of some of the landscapes was studied. The computed correlation functions are plotted in Figure 9.2 for the instances tai100a (left) and tai100b (right).

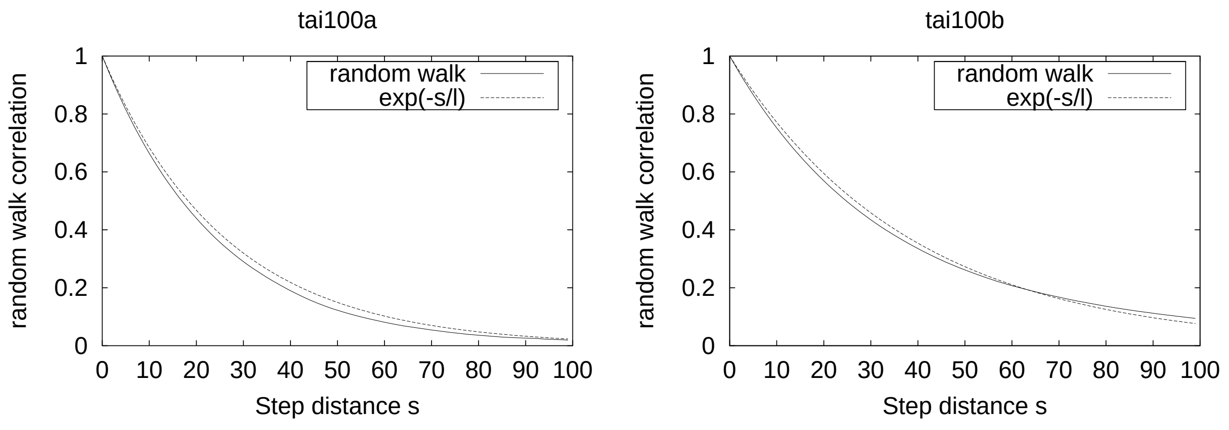


Figure 9.2: Random Walk Correlation functions for tai100a and tai100b

The random walk correlation functions have an exponentially decaying form as expected for AR(1) landscapes. The computed functions $r_c(s)$ are very close to the functions $r(s) = e^{-s/\ell}$, as can be seen in the plots. The latter function is included in the plots for comparison.

9.3.4 Fitness Distance Correlation Analysis

To investigate the distribution of local optima in the search space, 10000 local optima were produced using the *fast 2–opt* local search described above. In Table 9.1, the distance to the optimum or best-known solution $\langle d_{opt} \rangle$, the cost difference $\Delta c = c - c_{opt}$, the average quality $q = 100 \cdot (c_{opt}/c - 1)$ of the local optima, the fitness distance correlation coefficient ϱ, the mean distance between the local optima $\langle d_{ls} \rangle$, and the average number of iterations per local search i_{ls} are provided. Although for some instances more than one best-known solution may exist, only one best-known solution for each instance was considered in the experiments, since at most one best-known solution is available from QAPLIB. In Figure 9.3 the scatter plots (FDC plots) are provided for representatives of the studied instances.

For all studied instances, the distances between the local optima and best known solutions $\langle d_{opt} \rangle$ as well as the average distances between the local optima are very close to the diameter of the landscape. Thus, as can also be seen in the plots, the QAP instances have very unstructured landscapes. The local optima are neither restricted to a small region of the search space, nor do they seem to be correlated. An exception to this rule is instance tai100b, which exhibits a relatively high correlation of fitness and distance to the optimum. Surprisingly, the results can not be predicted by looking at the dominances of the instances. For example, the TSP instances and the GBP instances have a high value for $dom(A)$ and $dom(B)$, respectively, but do not show any correlation. Furthermore, instances tai80b and tai100b have similar flow and distance dominance values, but the former has significantly less correlated local optima. According to the distribution of the local optima, the instances can be divided into two types: the first type has correlated local optima and consists of the tai*b problems. The second type has no exploitable structure in the distribution of the local optima and consists of the remaining instances.

The Role of Epistasis and Structured Landscapes

To find out whether there are QAP landscapes that show a high correlation and have a correlation length close to $n/2$ and thus n/ℓ close to 2, new instances were created with varying epistasis. The two sets of problems consisting of 5 instances with $n = 100$ have been generated as follows. The distance matrix A of each instance was constructed by randomly creating n points in the unit square. An entry a_{ij} is defined by the Euclidean distance $D(i, j)$ between point i and j multiplied by 100. Thus, the entries of A lie between 0 and 100. The matrix B is constructed in a similar fashion by randomly creating n points in another unit square. For the first set of instances, b_{ij} is set to the Euclidean distance $D(i, j)$ multiplied by 100 between point i and j if the distance is below or equal to a predefined maximum distance D_{max}, zero otherwise. For the second set, b_{ij} is set to $100/(D(i, j) + 1)$, if $D(i, j)$ is below or equal to a predefined maximum distance D_{max}. Thus, by varying the threshold distance D_{max}, instances with arbitrarily epistasis/flow dominance can be created. The instances are structured since for the elements in the matrices the triangle inequality is obeyed. In real world applications at least one of the matrices represent a form of distance which naturally fulfills the triangle inequality. Thus, the generated instances are aimed to provide a structure that is found in real world applications. Table 9.2 provides an overview

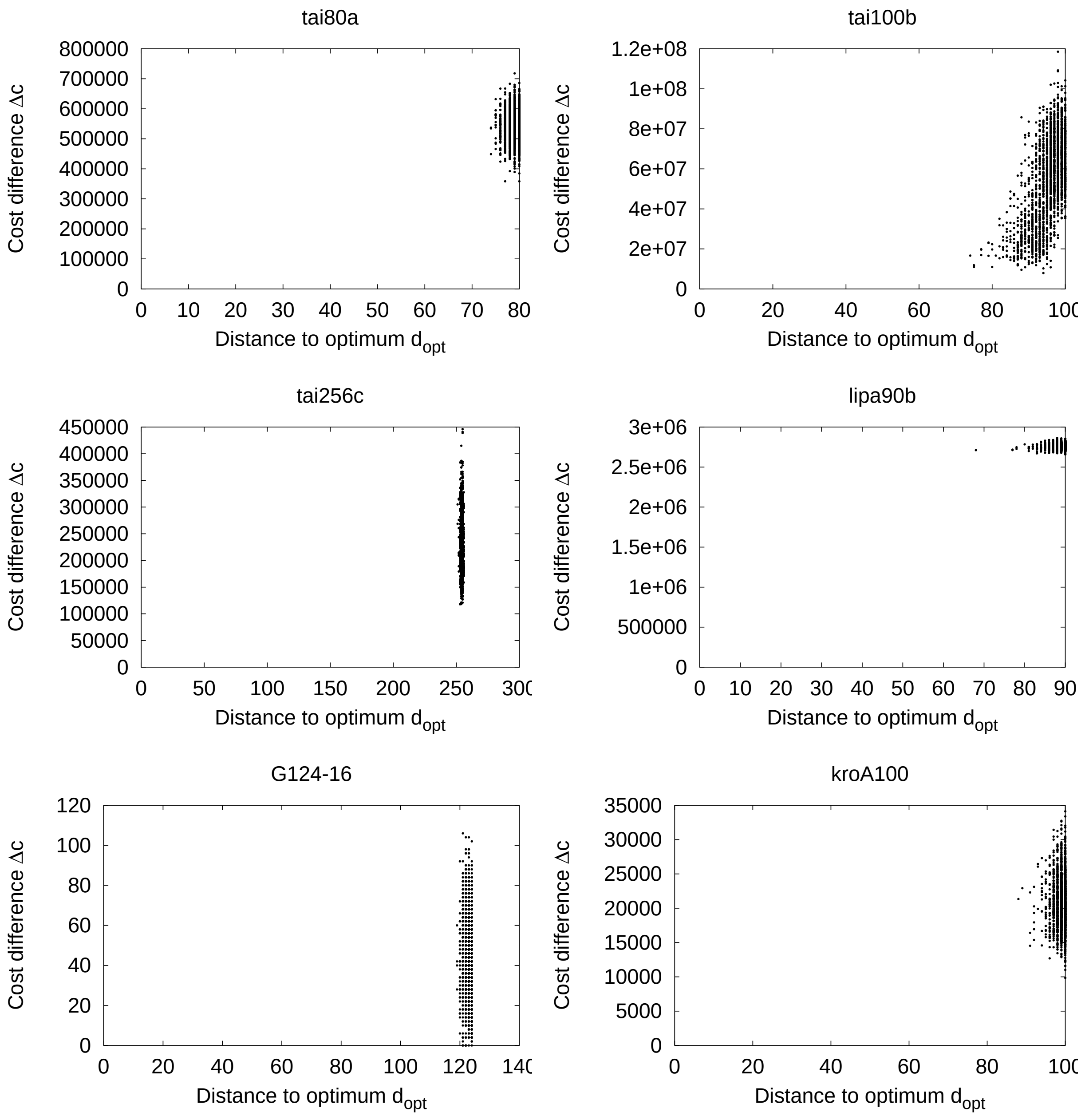


Figure 9.3: FDC analysis I

of threshold distances used. The same experiments were performed on the newly created

Table 9.2: Threshold parameters for the generated QAP instances

D_{max}	1	0.75	0.5	0.25	0.1
Set 1	pmd100a	pmd100b	pmd100c	pmd100d	pmd100e
Set 2	pmr100a	pmr100b	pmr100c	pmr100d	pmr100e

problems as for the previously described instances. The results are displayed in Table 9.3

and Figure 9.4. To obtain the best-known solutions for the new instances, several long runs (2 hours) with the MA using CX recombination as described below have been performed.

Table 9.3: Average distances and fitness distance coefficients for local minima

Instance	n	$dom(A)$	$dom(B)$	$\langle d_{opt} \rangle$	$\langle \Delta c \rangle$	q	ϱ	$\langle d_{ls} \rangle$	i_{ls}	n/ℓ	ℓ
pmd100a	100	48.2	50.9	93.9	49337.6	0.20	0.23	94.8	416.9	2.99	33.46
pmd100b	100	49.2	68.1	96.5	83625.5	0.54	0.26	97.0	450.5	2.97	33.67
pmd100c	100	48.9	119.9	93.4	115679.7	2.32	0.46	95.1	470.2	2.55	39.25
pmd100d	100	49.5	252.0	96.7	81714.1	17.80	0.29	97.6	446.9	3.10	32.30
pmd100e	100	49.3	686.7	98.0	5444.5	61.66	0.09	98.5	323.9	3.65	27.40
pmr100a	100	48.8	24.5	94.9	122075.6	0.40	0.24	95.5	427.9	2.42	41.27
pmr100b	100	48.4	52.4	95.1	192941.5	0.84	0.25	95.6	431.5	2.36	42.38
pmr100c	100	49.3	100.3	94.4	442360.6	3.62	0.42	95.8	447.9	2.50	39.95
pmr100d	100	48.8	237.9	97.3	497712.5	22.23	0.16	98.0	446.0	3.39	29.50
pmr100e	100	48.6	651.9	97.8	85554.9	67.60	0.15	98.5	341.6	3.73	26.82

The pmd* problems have a $dom(A)$ value of approximately 50, while the flow dominances $dom(B)$ are between 50 and 690. The pmr* instances have also a distance dominance of 50, but the flow dominance varies from 24 to 650. Again, the average distance to the best-known solution and the average distances between the local minima are close to the diameter of the landscape. The fitness distance correlation coefficient is significantly higher compared to the FDC of the other instances. However, all instances have a lower FDC than tai100b. pmd100c has the highest FDC ϱ in the first set, and pmr100c has highest FDC in the second. The plots in Figure 9.4 show that there exists a structure in the distribution of the local optima of the generated instances except for the ones with high flow dominance. Furthermore, it can be seen for pmd100c and pmr100c that there are local optima which are much closer to the best-known solution than most of the others. Even instance tai100b has no local optima which are similarly close to the best-known solution.

The correlation length ℓ in the first set is highest for pmd100c, and for the second set pmr100b has highest correlation length. The values for n/ℓ are closer to 2 than the value for tai100b. Neither the correlation length nor the FDC nor $\langle \Delta c \rangle$ reflects the effectiveness of the local search to approach the optimum cost: The quality q of the locally optimum solutions decreases rapidly with decreasing epistasis (increasing $dom(B)$). For example, for pmd100a, the locally optimum solutions have a cost 0.20% above the optimum on average, while for pmd100e the percentage excess is 61.71%! Thus, it seems that reaching the optimum cost is easy for meta–heuristics based on LS, if the instances are structured and there is high epistasis (low flow dominance). For instances with low epistasis (high flow dominance), reaching the optimum cost seems to be harder.

9.4 A Memetic Algorithm for the QAP

Although the memetic algorithm template is general in the sense that it can be used for every combinatorial optimization problem, some components are problem–specific. The creation of the initial population, as well as the the local search and the genetic operators are specific to the QAP, and will thus be described in the following.

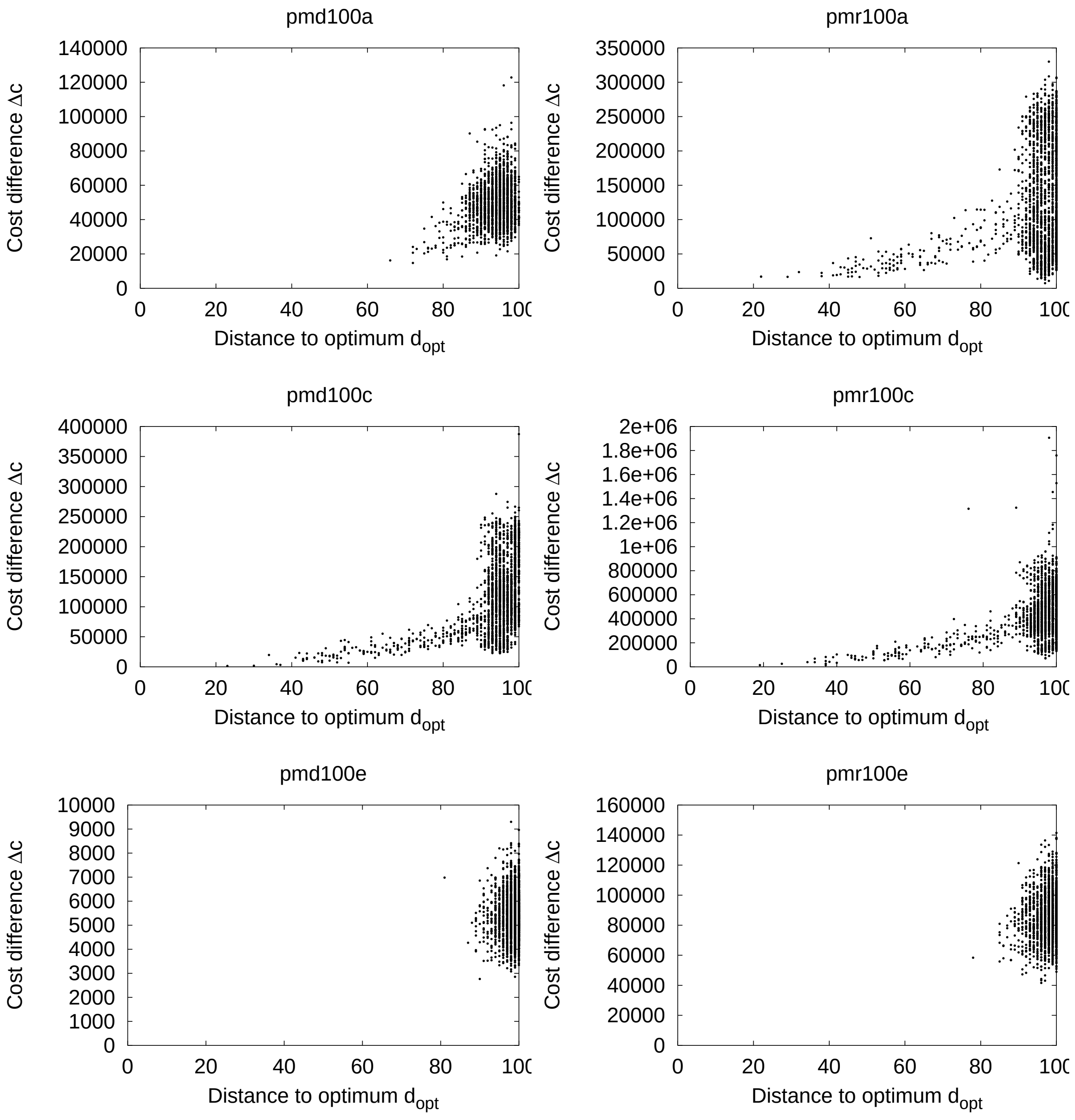


Figure 9.4: FDC analysis of additional QAP instances

9.4.1 Initialization and Local Search

The initial population of the MA is created by randomly generating solutions and applying a 2-opt local search. The solutions subject to the local search a generated in a poorly random fashion, since greedy heuristics or similar construction heuristics are not effective for most QAP instances.

The local search is a fast variant of *2-opt* accepting the first improving move found and making use of *don't look bits*. Equation (9.4) is not utilized since for each local search the gain matrix ΔC has to be initialized anew leading to an undesirable runtime penalty.

9.4.2 The Evolutionary Variation Operators

In the MA for the QAP described in this chapter, two recombination operators, the *distance preserving recombination operator* and the *cycle crossover*, as well as a simple mutation operator are employed each of which will be described in the following.

The DPX Recombination Operator

A distance measure between solutions is useful for explaining the behavior of evolutionary operators, since it defines the distance of the jumps made in the search space, and in some cases, the relative direction of a jump, if reference points exist. The evolutionary operators described in this section rely on the distance measure defined in equation (9.5).

The recombination operator DPX (*distance preserving crossover*) previously introduced in [209] relies on the notion of a distance between solutions. The basic idea behind DPX has also shown to be effective for the traveling salesman problem [105, 106], and can be described as follows. In general, the DPX is aimed at producing an offspring which has the same distance to each of its parents, and this distance is equal to the distance between the parents themselves. The alleles that are identical for the same genes in both parents will be copied to the offspring. The alleles for all other genes change. Thus, although the crossover is highly disruptive, the local search procedure applied subsequently is forced to explore a region of the search space that is defined by the genes with different alleles in the parents, which is the region where better solutions are most likely to be found in some search spaces.

The DPX operator for the QAP works as follows. Suppose that two parents A and B as shown in Figure 9.5 are given.

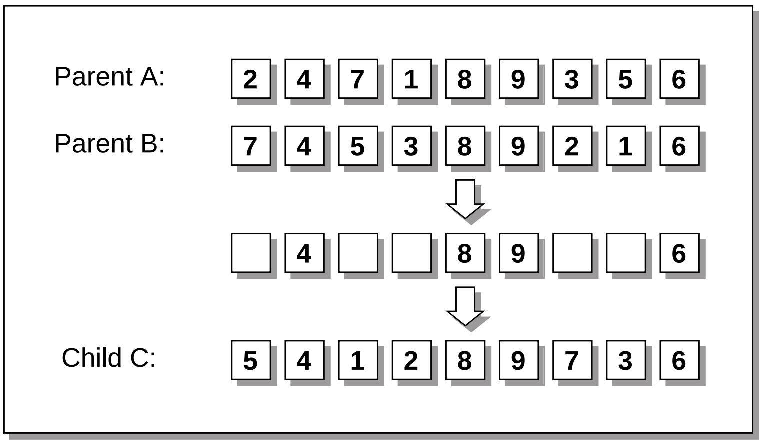


Figure 9.5: The DPX Recombination operator for the QAP

First, all assignments that are contained in both parents are copied to the offspring C. The remaining positions of the genome are then randomly filled with the yet unassigned facilities, taking care that no assignment that can be found in just one parent is inserted into the child. After that, we end up with a child C, for which the condition $d = d(C, A) = d(C, B) = d(A, B)$ holds; in the example in Figure 9.5 the distance d is 5. Since it is computationally expensive to ensure that the child has exactly the distance d to both parents, a linear time algorithm is used that produces offspring that are with high probability d units away from their parents. Experiments have shown that there is no significant performance difference between the exact and approximate version of DPX in terms of solution quality. Therefore, we prefer the much simpler approximate algorithm.

The CX Recombination Operator

The second recombination operator (CX) recently proposed for the QAP [213] – the name was changed from UX to CX to account for the fact that it is similar to the cycle crossover proposed for the TSP – preserves the information contained in both parents in the sense that all alleles of the offspring are taken either from the first or from the second parent. In other words, the operator does not perform any implicit mutation, since a facility that is assigned to location i in the child is also assigned to location i in one or both parents. The CX recombination operator works as shown in Figure 9.6.

```
function CXrecombination($a \in \Pi(n)$, $b \in \Pi(n)$) : $\Pi(n)$;

  begin
    mark all $c(i)$ as unset;
    set $c(i) := a(i)$ for all $i$ where $b(i) = a(i)$;
    set $j$ to a random position between 0 and $n - 1$ with $c(j)$ unset;
    while not all locations of $c$ are set do
      assign $a, b$ randomly to $parent, other$;
      $l := j$;
      repeat
        $c(l) := parent(l)$;
        $k := l$;
        find $l$ with $parent(l) = other(k)$;
      until $other(k)$ in $c$;
      $j := j + 1 \mod n$;
    endwhile;
    return $c$;
  end;
```

Figure 9.6: The CX recombination

In the first phase, all facilities found at the same locations in the two parents are assigned to the corresponding locations in the offspring. Then, starting with a randomly chosen location with no assignment, a facility is randomly chosen from the two parents. After that, additional assignments are made to ensure that no implicit mutation occurs. Then, the next unassigned location to the right (in case we are at the end of the genome, we proceed at its beginning) is processed in the same way until all locations have been considered.

Consider the example shown in Figure 9.7. First, all facilities that are assigned to the same location in the parents are inherited by the offspring. These are the facilities 4, 9, and 6. Then, beginning with a randomly chosen location (in this case location/position 3), a facility is randomly selected from one of the parents and is assigned to the same location in the child. In the example, this is facility 7. Now, other facilities have to be assigned to guarantee that no implicit mutation occurs. By assigning facility 7 of parent A to location 3

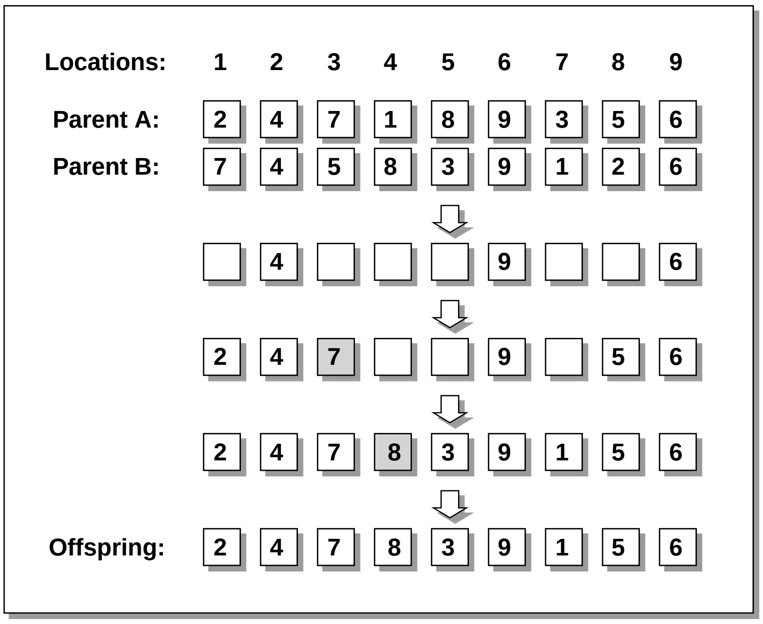


Figure 9.7: The CX recombination operator for the QAP

we prevent the possibility to assign facility 5 of parent B to that location. Hence, we have to assign facility 5 to the same location as in parent A. Again, assigning facility 5 to location 8 requires that facility 2 has to be assigned, too. After that, facility 2 is located at site 1 in the genome. Since the facility at location 1 in parent B is 7, and 7 is already included in the child, we can proceed in choosing a facility for the next free location to the right in the offspring. In the example, facility 8 of parent B is inserted into the offspring, and to avoid implicit mutations, we have to insert facility 1 at location 7 and facility 3 at location 5. Then, the algorithm terminates since all facilities have been assigned.

To limit the region where the local search takes place after the DPX or CX recombination operator has been applied, the genes with the same alleles contained in both parents are fixed. Hence, in the above example, swaps can only be performed between locations 1, 3, 4, 5, 7, and 8. This restricts the local search to the subspace defined by the two parents. The neighborhood size for the local search is reduced from $|\mathcal{N}_{2opt}| = \frac{1}{2}n\cdot(n-1)$ to $|\mathcal{N}| = \frac{1}{2}d\cdot(d-1)$ (with $d = d(\pi_1, \pi_2)$), which results in an increased performance of the local search phase, since the average distances between individuals of the population decrease during the evolution.

The Mutation Operator

The mutation operator used in the MA approach exchanges a sequence of facilities in the solution until the offspring has a predefined distance to its parent, as shown in Figure 9.8. To ensure that the offspring has the predefined distance, in each step, the second facility chosen is exchanged again in the succeeding step, such that the resulting distance between parent and offspring is one plus the number of exchanges. To illustrate the operation of the mutation operator, consider the example shown in Figure 9.8.

In the first step, facilities 9 and 4 are exchanged, then facility 4 and 1, and in the last step facility 1 is exchanged with 3. Thus, the resulting (mutation jump) distance between parent and offspring is 4.

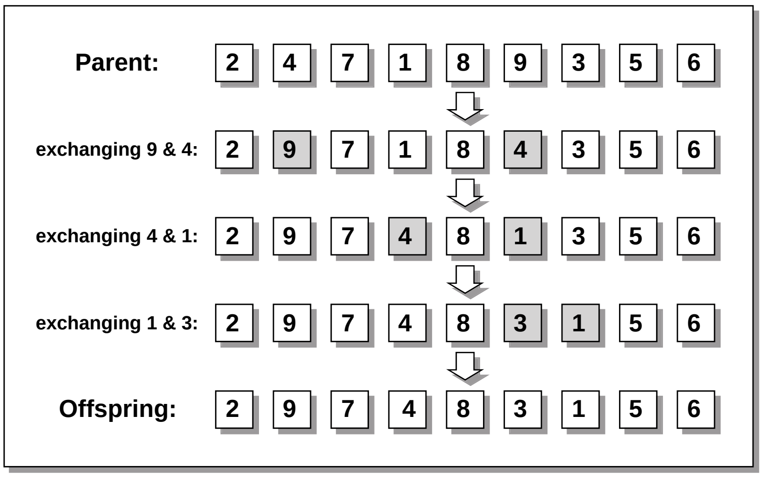


Figure 9.8: The Mutation Operator for the QAP

9.5 Memetic Algorithm Performance

In the first experiment, the recombination operators DPX and CX described above as well as the mutation operator with several jump distances were tested. Runs were performed on the instances tai100a, tai100b, tai150b, tho150, tai256c, kroA100, and all pmd* and pmr* instances. Results are displayed for selected instances in Figure 9.9.

The experiments show that for all instances except tai100a and tai256c, the CX operator is more effective than the mutation operator. Furthermore, the DPX recombination operator is outperformed by CX for all tested instances. In case of tai100a and tai256c, mutation becomes superior to recombination for some jump distances. These instances have a n/ℓ value close to 4 and are considered to be unstructured. Although kroA100 has an extremely rugged landscape ($n/\ell \sim 4$), the MA with CX recombination is better or as good as the MAs using mutation. The DPX recombination shows to be efficient only for the instances tai100b and pmr100b.

In the studies, a general rule for setting the optimum mutation jump distance could not be found. Besides the fact that the optimum depends on the structure of the landscape (e.g. the optimum for tai100a is 20 while for tai100b it is 40), the running times of the algorithms appear also to have an influence. However, it seems that the optimum jump distance lies below the correlation length of the landscape (below $n/4$) for unstructured landscapes.

The results can be summarized as follows. If the landscape is highly rugged ($n/\ell \sim 4$) and the average number of improvements (iterations) made by the local search is low, recombination becomes inefficient and mutation with a jump distance below the correlation length of the landscape is the best choice. DPX recombination is only efficient if the landscape is highly structured (low n/ℓ and correlation between local optima). CX recombination is the best choice for all landscapes with low n/ℓ and for rugged landscapes ($n/\ell \sim 4$) with low epistasis (high flow or distance dominance).

9.5.1 Comparison of Heuristic Algorithms for the QAP

To evaluate the performance of the MA using CX relative to other proposed approaches for the QAP, a comparison study with 5 of its competitors was performed. Included in

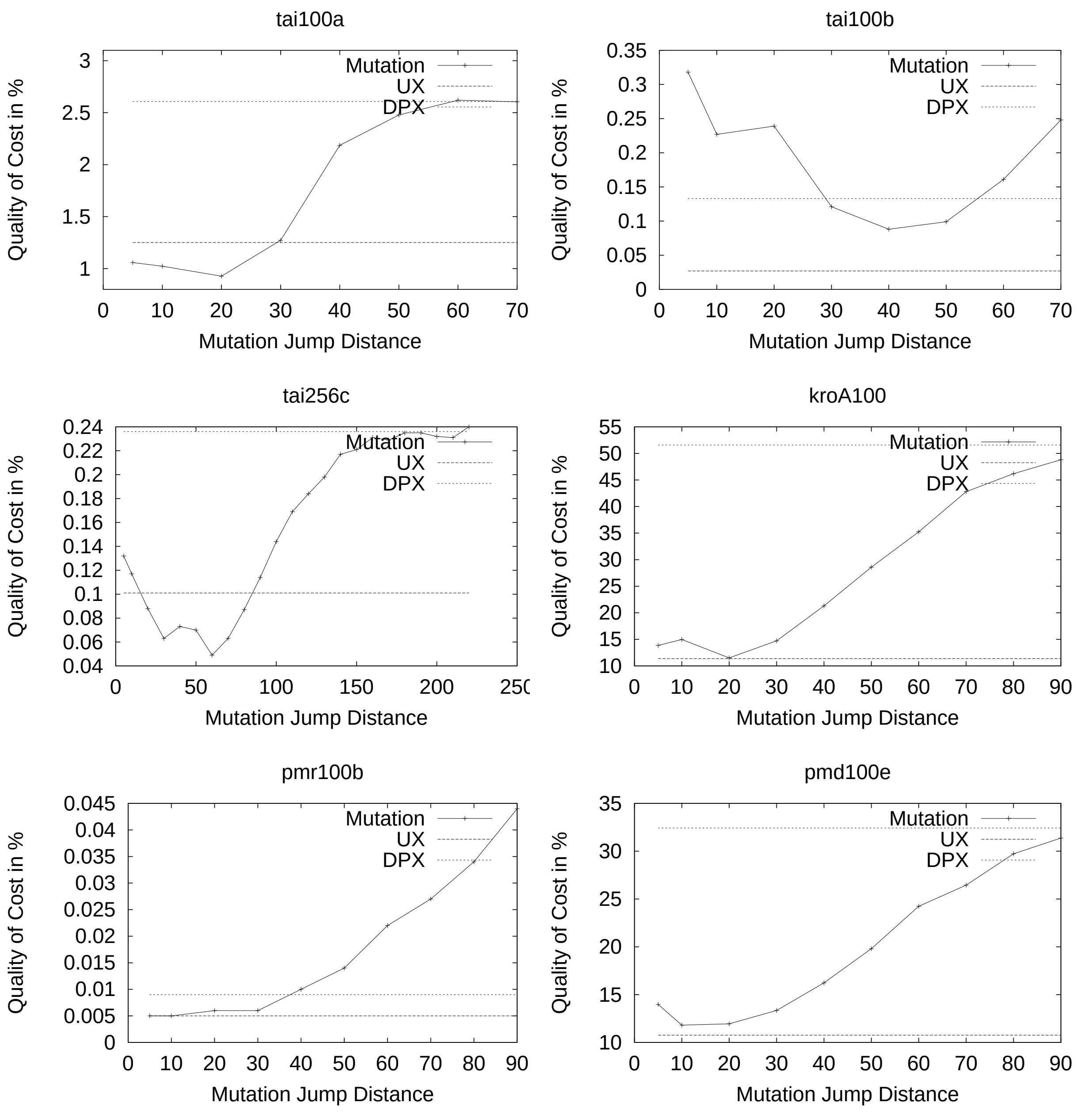


Figure 9.9: Operator performance for tai100a, tai100b, and tai256c

the comparison are the *robust tabu search algorithm* (Ro-TS) [295], the *reactive tabu search algorithm* (Re-TS) [23], the *fast ant colony algorithm* (FANT) incorporating local search [298, 297], *simulated annealing* (SA) according to Connolly [56], and the Min-Max Ant System (MMAS) [286, 290].

16 QAP instances from the QAPLIB were selected, ranging from $n = 19$ locations up to $n = 256$. Included are structured as well as unstructured instances.

Figure 9.4 displays the results obtained by the MA, together with the results produced by the robust tabu search algorithm (Ro-TS), reactive tabu search algorithm (Re-TS), fast ant colony algorithm (FANT), simulated annealing (SA), and the min–max ant system (MMAS). The five competitors belong to the best currently available heuristic approaches to the QAP.

To enable a fair comparison with the MA, the code developed by the corresponding authors was used and executed the programs on the same hardware and operating system platform. All algorithms were terminated after the same predefined time for each instance.

Table 9.4: Comparison of four algorithms for the QAP

		MA$_{R=3}$	**MA**$_{R=1}$	**Ro-TS**	**Re-TS**	**FANT**	**MMAS**	**SA**	
instance	best-known	avg.%	avg.%	avg.%	avg.%	avg.%	avg.%	avg.%	t/s
els19	17212548	• 0.000	• 0.000	0.124	• 0.000	• 0.000	• 0.000	31.96	5
chr25a	3796	1.222	• 0.000	6.567	5.297	4.549	*0.669	49.52	15
bur26a	5426670	0.003	• 0.000	0.001	0.029	0.020	• 0.000	0.390	20
nug30	6124	0.004	0.007	0.020	0.589	0.219	*• 0.000	0.882	20
kra30a	88900	0.369	• 0.000	0.223	0.612	0.931	0.119	4.601	20
ste36a	9526	• 0.045	0.091	0.128	1.189	0.545	*0.051	12.55	30
tai60a	7208572	1.314	1.597	1.313	• 0.794	2.577	*1.159	3.199	90
tai80a	13557864	1.106	1.305	1.023	• 0.482	2.525	*0.768	3.298	180
tai100a	21125314	1.089	1.252	0.909	• 0.385	2.569	*0.728	1.848	300
sko100a	152002	• 0.096	0.127	0.191	0.397	0.474	*0.195	2.942	300
tai60b	608215054	• 0.000	• 0.000	1.898	0.929	0.213	0.075	1.760	90
tai80b	818415043	0.191	• 0.004	2.929	1.602	0.821	0.718	5.092	180
tai100b	1185996137	0.076	• 0.038	2.373	1.469	0.360	0.328	6.696	300
tai150b	498896643	• 0.361	0.397	2.851	1.775	1.176	1.167	3.787	600
tho150	8133484	• 0.151	0.202	0.548	0.488	0.765	*0.395	2.939	600
tai256c	44759294	0.070	0.099	0.326	0.266	0.273	*0.067	0.370	1200

The MA was run in two variants, one with a diversification rate $1/R$ of 1, and one with a diversification rate of 1/3. In the table, *instance* denotes the name of the QAP instance from the QAPLIB (the number indicates its size n). The best-known solution is provided for each instance in the second column. For each algorithm, *avg.* shows the average percentage excess over the best-known solution obtained within 30 runs, and *t/s* displays the time in seconds allowed for each of the 30 runs. The dots in each row indicate the best average performance for each instance. The min–max ant system was run in two variants, one incorporating 2–opt local search and the other using tabu search. The results for the best of the two variants are displayed in the column denoted MMAS. An asterisk indicates that the result was obtained by the tabu search variant.

The results indicate that the MA is superior, in terms of solution quality within a given time limit, to all alternative approaches for all but five instances. The unstructured instances tai60a, tai80a, and tai100a, are solved more effectively by Ro-TS, Re-TS and MMAS, but these instances have no practical importance. For instances nug30 and tai256c, MMAS shows a slightly better performance than the MA. However, the MA using the mutation operator finds the best-known solution of tai256c 8 out of 30 times with an average percentage excess of 0.038%, and thus outperforms MMAS. For the problems tai60a, tai80a, and tai100a, the results of the mutation based MA are 1.385%, 1.001%, and 0.936%, respectively. In an additional experiment, the MA with CX has been shown to be superior to an MA with of combination of CX and mutation for the structured problems.

In fact, it is hardly surprising that the MA is not the top performer for the three uniformly randomly generated instances (tai60a, tai80a, and tai100a), since the MA is designed to exploit some kind of (assumed) structure in the QAP search space; if there is no structure at all, there is not much the MA can do (except for randomly jumping around). However, considering that the performance of the MA for tai60a, tai80a, and tai100a is still acceptable (better than FANT and nearly as good as Ro-TS), and the MA outperforms its competitors on the remaining instances, the MA appears to be the method of choice for the QAP instances studied.

An explanation why the tabu search algorithms and the MMAS with tabu search perform better than MMAS with 2-opt local search and the MA can be found with a little help from the fitness landscape analysis. For the instances tai60a, tai80a, and tai100a, the average number of iterations of the local search is low. Tabu search allows to find much better solutions since it does not stop in a local optimum. It efficiently searches for other improvements once a local optimum has reached. For totally unstructured landscapes, this appears to be the best strategy.

To illustrate the behavior of the MA using CX in more detail, Table 9.5 shows the times (of 30 runs) required by the MA to reach the best-known solution. In the figure, *gen* denotes the number of generations, *average* the average value of the objective function (equation (2.11)), *deviation* the average deviation from the known optima, *min. t in s* and *avg. t in s* the minimum and average runtime in seconds, respectively. The results illustrate that the algorithm is able to find the best-known solutions in *all* 30 out of 30 runs in short average running times for all structured problems.

Table 9.5: Shortest and average times to reach the best-known solution with the MA

instance	gen	average cost	quality	min. t in s	avg. t in s
chr25a	131	3796.0	0.00%	0.3	2.6
bur26a	20	5426670.0	0.00%	0.2	1.0
nug30	234	6124.0	0.00%	0.8	7.1
kra30a	83	88900.0	0.00%	0.4	2.7
ste36a	925	9526.0	0.00%	2.0	36.7
tai60b	151	608215054.0	0.00%	7.5	23.2
tai80b	787	818415043.0	0.00%	54.8	258.3
tai100b	1312	1185996137.0	0.00%	57.5	629.1
G124-02	12	26.0	0.00%	9.2	33.2
G124-16	2	898.0	0.00%	7.4	15.0
lipa90b	121	12490441.0	0.00%	5.2	72.1
pmd100c	58	4992484.0	0.00%	19.4	41.1
pmr100c	143	12223044.0	0.00%	21.1	72.0

Although the alternative approaches used in the comparison are – to the best of the authors' knowledge – the most powerful methods available today for solving QAP instances, several other techniques have been published. For example, another high quality algorithm based on the ant system, called HAS-QAP [111], has been proposed which is not included in the comparison, since the source code was not available to us. However, the MMAS algorithm appears to be superior to HAS-QAP when selecting the best of MMAS with 2-opt and MMAS with tabu search [288]. The results presented in [111] indicate that the MA is superior to HAS-QAP in terms of CPU time and solution quality for problems with $n > 30$.

Compared to further approaches, it seems that the proposed MA algorithm is also superior. In [231], the parallel genetic algorithm described above has been used to solve the instances ste36a and nug30. The optima for both problems could be found in 279 seconds and 363 seconds, respectively, on a transputer with 64 processors. However, on a state-of-the-art workstation, the optima could be found in less than 3 seconds by the proposed MA. In [199], results are reported for solving nug30 with the *Ant System* (AS). The best solution found by a combination of AS and simulated annealing had a fitness value of 6128 while the MA finds the optimum solution with a value of 6124 easily. The *combinatorial evolution strategy (CES)* proposed in [237] produces remarkable results for problems of up to size 64 without any kind of domain knowledge, but the results are worse in both quality and computation time than the results presented here. In [300], a genetic algorithm has been proposed and tested for problems of up to size 30. For the largest problem investigated there (nug30), the optimum could not be found. Comparison studies have been made in [14, 200] for relatively small instances with the conclusion that genetic algorithms perform relatively poor, but that hybridization may enhance the search significantly. The Genetic Hybrids [93] have been proven to be effective for finding near optimum solutions: In long runs (up to 22 hours on a Sun Sparcstation 10) new best solutions could be found for some of the problems contained in QAPLIB. In comparison studies [296, 298], the genetic hybrids have been shown to be the best algorithms for most of the studied instances. However, the results presented in [298] are worse than the results obtained with the MA on the structured tai*b problems in both quality and running times.

9.6 Summary

In this chapter, the fitness landscapes of several instances of the quadratic assignment problem (QAP) have been analyzed. To investigate the local and global properties of the fitness landscapes, an autocorrelation analysis and a fitness distance correlation analysis has been utilized as described in chapter 4. While the local properties have a large impact on the effectiveness of a local search algorithm, the global structure can be exploited by a population based search. Thus, the analysis is especially important for the design of hybrid algorithms.

The analysis has revealed that the majority of problems is unstructured with respect to their global and local structure. Many instances represent highly rugged landscapes and the local minima are totally uncorrelated; they appear to be uniformly distributed over the search space. Furthermore, it has been shown that low dominance proposed as a measure for problem difficulty for the QAP is not suited to predict the hardness of a problem instance since it does not reflect the structure in a fitness landscape. However, with flow and distance dominance, a measure for gene interaction (epistasis) can be defined which allows to separate instances of the QAP, if the correlation length of the landscape as a measure the landscape ruggedness, and the correlation of fitness and distance of the local minima in the landscape, is also considered.

To investigate the influence of epistasis on the structure of the fitness landscapes, structured instances with varying flow dominance and thus varying epistasis were generated. It has been shown that there are instances exhibiting highly structured, rather smooth landscapes in the QAP besides the ones obtained from QAPLIB, a public library for QAP instances.

Epistasis, correlation length and fitness distance correlation considered together, allows to separate QAP instances into four classes. The first type, characterized by high epistasis, high landscape ruggedness (low correlation length) and uncorrelated local optima, is the hardest of all types of instances. Both local search and population based meta–heuristics are relatively ineffective since there is no local or global structure to exploit. The second type of instances has a rugged landscape but low epistasis and no correlation between fitness and distance of the local optima. These instances are considered to be easier to solve but still are very unstructured. The instances that are easiest to search have a relatively smooth landscape (high correlation length) and the local optima are correlated. Instances with a relatively smooth landscape, but very low epistasis and no global correlation structure constitute the fourth class.

A memetic algorithm (MA), i.e. a genetic algorithm incorporating 2-opt local search was applied to instances of the four classes with recombination or mutation operators. For, the first and hardest class, it turned out that recombination is ineffective, which is not surprising since recombination is thought of as exploiting a present structure in a problem, and for these problems the structure is missing. For all other types of instances the MA with the recently proposed recombination operator CX performed best. A highly disruptive form of recombination as realized in the DPX recombination operator appears to be only effective for the instances with correlated local optima and thus for the class of problems with exploitable global structure.

Thus, it has been shown how a fitness landscape analysis is useful for finding a suitable choice of evolutionary operators in an MA for the QAP. However, the fitness landscape analysis may also be used for other meta–heuristics such as the ant colony system: The best choice of local search or tabu search can be easily done based on a landscape analysis.

To compare the performance of the memetic algorithm, a comparison study was conducted on a set of QAP instances with a problem size up to $n = 256$. In particular, the MA was compared with five very good alternative heuristic approaches to the QAP: *reactive tabu search* [23], *robust tabu search* [295], *fast ant colony system* [298], the *min-max ant system*, and *simulated annealing*. The MA using the CX recombination operator is shown to be superior to the other approaches, except for the hardest, totally unstructured instances. Furthermore, the MA approach proves to be very robust, since the best-known solutions could be found in *all* runs with short average running times for instances with a problem size up to $n = 100$.

Chapter 10

Conclusions

The research in the field of combinatorial optimization is primarily focused on developing new or improved heuristics for selected combinatorial problems. Not much effort has been made in gaining insight into the structure of such problems in order to predict the performance of existing search strategies or to develop even more effective approaches. Therefore, this work is intended to show not just *that* a given search strategy is effective for some problems. The main goal is to find reasons *why* a search strategy in a given situation performs better than others.

The studies in this thesis concentrated on a particular class of heuristics: memetic algorithms (MAs). The combination of efficient neighborhood search strategies and evolutionary algorithms has been shown to be highly effective for many combinatorial optimization problems. While the neighborhood search is responsible for finding locally optimum solutions in small regions of the search space, the evolutionary component is responsible for exploring new promising regions. Each of the two components requires different problem characteristics in order to be effective. These characteristics are best described with the notion of fitness landscapes: if the candidate solutions of an optimization problem are arranged in a spatial structure, and the heights of these points are determined by the solution quality (fitness), they are said to form a fitness landscape.

The effectiveness of neighborhood search methods is strongly related to the ruggedness of the landscape, i.e., the correlation of the fitness of neighboring points in the search space. A measure for landscape ruggedness is the correlation length of a landscape that can be either calculated mathematically or estimated experimentally employing an autocorrelation or random-walk correlation analysis. On the other hand, the effectiveness of evolutionary variation is strongly influenced by the distribution of the points produced by the neighborhood search. If there is a structure in the distribution of these points, i.e., there is correlation between the fitness of the points and their distance to the highest peak in the landscape, the evolutionary component can be designed to exploit this feature. A fitness distance correlation (FDC) analysis can be utilized to discover such a present global structure.

After an introduction to combinatorial optimization problems, memetic algorithms, and fitness landscapes in the chapters 2 through 4, five different combinatorial optimization problems have been investigated in the chapters 5 through 9: *NK*-landscapes, unconstrained binary quadratic programming (BQP), the traveling salesman problem (TSP), the graph bipartitioning problem (GBP), and the quadratic assignment problem (QAP). The local structure in terms of landscape ruggedness and the global structure in terms of the distribution of the local optima produced by a local search have been investigated. For three out of the five considered problems, an autocorrelation analysis has been performed by other researchers and thus a theoretical value for the correlation length exists. For the unconstrained binary quadratic programming problem and the quadratic assignment problem, a random-walk analysis has been performed experimentally in this work. For all five problems, a fitness distance correlation analysis has been conducted.

Furthermore, new evolutionary variation operators and/or greedy and local search heuristics have been proposed for memetic algorithms for all problems: For *NK*-landscapes, a new greedy heuristic, a new *k-opt* local search, and a new greedy recombination operator have been proposed in chapter 5. In chapter 6, a new greedy heuristic, and a new *k-opt* local search have been introduced for binary quadratic programming. For the traveling salesman problem, two new recombination operators have been proposed: a distance preserving recombination operator and a generic greedy recombination operator. Both operators are described in chapter 7. A new greedy recombination operator could be successfully applied for graph bipartitioning in chapter 8. For the QAP, two new recombination operators have been described in chapter 9: a distance preserving recombination operator and a cycle recombination operator similar to the cycle crossover for the TSP. In all five cases, the memetic algorithms employing the new operators and/or local search and greedy heuristics are shown to be among the most efficient heuristics available.

From the various landscape analysis experiments some important conclusions can be drawn as described in the following paragraphs.

Conclusions drawn from the fitness landscape analysis

The properties of fitness landscapes can vary enormously between problems and even between instances of the same problem. A fitness distance correlation analysis has been performed by researchers in their studies for some of the investigated problems. However, it has been shown in this thesis that investigating a single or few instances is not sufficient for drawing general conclusions: The instances of the graph bipartitioning problem investigated in chapter 8 are selected to cover all available types of graphs. In fact, the results of the FDC analysis vary drastically between the types of graphs. There are highly correlated landscapes and landscapes with no correlation of fitness and distance to the optimum.

Regular instances and instances for which optimum solutions can be calculated in polynomial time may not reflect the characteristics of real-world or complex instances. For the traveling salesman problem, the graph bipartitioning problem, and the quadratic assignment problem it has been shown that generated instances with known optimum solutions (the optimum solution can be calculated in polynomial time) are not well suited for the evaluation of heuristics since they do not resemble characteristics of the instances with higher computational complexity. The fractal instances of the traveling salesman problem, the regular graphs in the graph bipartitioning, and the quadratic assignment problem instances generated with the Li-Pardalos generator mentioned in chapter 9 are such examples.

Random instances may be not as hard as real-world instances. A uniform distribution of instance data may not resemble the structure of real-world instances. The results obtained from the fitness landscape analysis indicate that random benchmark instances for combinatorial problems should be generated with care. In the TSP, choosing randomly generated instances with points uniformly distributed in the unit square has the advantage that a tight lower bound on the optimum tour length exists. However, it has been shown in the analysis in chapter 7 that these instances show a higher correlation of fitness and distance to the optimum than other instances. In the graph bipartitioning problem, it appears to be crucial in which way random graphs are generated. If the edges of a complete graph are selected with a uniformly with a given probability, the resulting graphs have totally different landscape characteristics compared to random graphs in which all edges connect vertices that are closer or or equal to a predefined Euclidean distance in a two-dimensional plane. The latter type of graph more appropriately resembles the structure of graphs found in applications of the GBP. In the quadratic assignment problem, the influence of random choices becomes even more obvious. Randomly generated flow and distance matrices with uniformly distributed element values yield totally uncorrelated landscapes, while instances with randomly generated matrices based on distributions found in real-life problems show correlation of fitness of the local optima and their distance to the optimum. *NK*-landscapes have fitness contribution functions with a uniform distribution in contrast to binary quadratic programs. Consequently, the landscape properties of the two problems differ significantly. Thus, uniformly distributed instance data may probably not be appropriate for generating realistic benchmark instances for combinatorial problems.

Test instances should be chosen to cover all types of fitness landscapes. A diverse set of random instances can be generated taking the fitness landscape properties into account. In the QAP, it has been shown that creating matrices with distributions of the element values likely to be found in real-world applications, instances with arbitrary landscape ruggedness can be generated.

The size of the region in which the local optima are contained matters. In the studies, another aspect besides fitness distance correlation has been found to be important: the size of the region in the search space containing the local optima. For some problems like most instances of the QAP, the local optima are distributed over the whole search space while for other problems like the TSP they are contained in a small fraction of the search space. If this fraction is very small as in the case of binary quadratic programs, this property can be exploited by heuristics.

High epistasis does not always imply hard search. Epistasis – the amount of interaction of the components in a solution – in a problem has been widely recognized as an indicator for problem hardness. In the *NK*-landscape model, increasing epistasis yields increasing landscape ruggedness and a higher number of local optima. This rule appears to be specific to *NK*-landscapes. For some binary quadratic programs, landscape ruggedness decreases slightly with increasing epistasis, as shown in chapter 6. Moreover, in the studies of the graph bipartitioning problem, it could be observed that increasing epistasis can yield a decreasing number of local optima. More important than just the number of interacting solution components appears to be the topology of interactions. In chapter 8, the notion of a dependency graph has been introduced defining which solution components influence the fitness contribution of other components. The structure of the dependency graph has been shown to be a better indicator for problem hardness; in some cases it appears not to be sufficient to take just the degree into account. The way in which random instances are generated directly influences the dependency graph. Uniform distribution of instance data as discussed above certainly yields different dependency graphs than non-uniform distribution.

Performance prediction requires a deep understanding of the problem under consideration. The experiments have shown that the performance of heuristics cannot be predicted based on problem characteristics alone without taking the behavior of the considered heuristics into account. In experiments on the QAP, it turned out that dominance, a measure of epistasis for the QAP, and the results of a fitness landscape analysis together are required to predict the performance of a memetic algorithm. The studies have shown that performance prediction is a complicated task and is only reasonable if all or most problem characteristics having influence on heuristic search are taken into account. Today, not much about these characteristics is known. It is very likely that other yet undiscovered characteristics are important or even dominant in some cases. Nevertheless, the local structure of the landscape – its ruggedness – and its global structure in terms of fitness distance correlation appear to be key characteristics in memetic search. Since the fitness landscape is a product of the central component of the memetic algorithm, the fitness landscape analysis provides algorithm specific results compared to the dominance measure noted above.

Conclusions drawn from memetic algorithm experiments

Several conclusions could be drawn from the experiments performed with the proposed memetic algorithms. These conclusions are described in the following.

Memetic algorithms scale much better with the problem size than evolutionary algorithms or multi–start local search. Multi-start local search or simple evolutionary algorithms perform significantly worse than the combination of local search and evolutionary algorithms, as shown in various experiments in chapters 5 through 9. In comparison to the latter, memetic algorithms have been shown to scale significantly better with the problem size even for problems with ideal properties for the application of GAs/EAs (binary representation and no constraints) like *NK*-landscapes and binary quadratic programs. The inferior performance of evolutionary algorithms based on the recombination of solutions in permutation problems like the TSP is thus only partially explained by the difficulty of satisfying the implicit constraints of the problem.

Greedy components in a memetic framework are promising for problems of low epistasis. Greedy components can be incorporated into the initialization phase of a memetic algorithm as well as in the recombination step. For problems with low epistasis this is a very promising approach, as shown for the traveling salesman problem, *NK*-landscapes with low epistasis (low values of K), and graph bipartitioning problem instances with low vertex degree.

Memetic algorithms are most effective on correlated landscapes. The most important property of landscapes on which MAs have been shown to be highly effective is the correlation of fitness of the local optima and their distance to the global optimum. MAs employing recombination are capable of exploiting this structure since with respectful recombination, offspring are produced located in a subspace defined by the parents. These offspring are used as starting solutions for a local search that likely ends in a local optimum relatively near the starting point and thus near the parents. Due to the correlation structure of the search space this point has more likely a higher fitness than local optima more distant in an arbitrary direction from both parents. Viewing the evolutionary search process as a whole, recombination is capable of decreasing the distance between the solutions (and the distance to the optimum) if the landscape is correlated, while fitness is increased by selection: the higher the fitness of the local optima and the closer they are in terms of distance, the more likely the optimum is found in the vicinity of these solutions.

Memetic algorithms should employ respectful recombination operators on correlated landscapes. In experiments on the traveling salesman problem, it turned out that the most important property of a recombination operator is respectfulness, i.e., all edges (solution components) common to both parents are also found in the offspring. Other properties had only minor influence due to the effects of the local search. In cases where the landscapes are unstructured (exhibit no correlation) recombination is ineffective and a simple mutation that jumps out of the basin of attraction of the current local optimum proves to be more suitable, as shown on the totally unstructured landscapes of the quadratic assignment problem or the *NK*-landscapes with large K. For these landscapes, other techniques might be more effective, such as sophisticated tabu search variants that incorporate mechanisms to explore unvisited regions once a region has exhaustively been searched without searching the same region twice. However, it is believed that such unstructured landscapes most likely do not appear in real-world applications if the best suited neighborhood structure for the landscape is chosen.

If the local optima are correlated, but too close to each other, recombination becomes ineffective. The experiments on the binary quadratic programming problems have shown that high correlation of local optima does not always guarantee a good performance of recombination-based MAs. The reason lies in the fact that the local optima of the studied problem instances are too similar. A recombination too often yields a starting solution for which the local search ends in a solution equal to one of the parents. In this case, a memetic algorithm with a mutation operator producing offspring with a constant distance to the parents has been shown to be more effective.

Iterated Local Search is highly effective in the TSP but inferior to population-based search for other combinatorial problems. For the TSP, an effective combination of a simple mutation and a sophisticated local search exists – the non-sequential four-change and the Lin-Kernighan heuristic. This combination has a similar performance as recombination-based memetic search. However, such a combination is believed not to exist for other combinatorial problems - the high effectiveness of iterated local search appears to be limited to the TSP.

Areas of future research

There are several areas of future research. In this thesis, two classes of combinatorial optimization problems have been addressed: Problems without implicit or explicit constraints and problems with implicit constraints according to the classification introduced in chapter 2. Combinatorial problems with explicit constraints are probably more common in real-world applications. The research presented in the thesis is essential for the analysis of these problems since the problems addressed here can be regarded as special cases of the class of combinatorial problems with explicit constraints. This work is worth being continued to cover all kinds of combinatorial problems with explicit constraints and with or without implicit constraints.

Memetic algorithms require some parameters such as population size and the number of generated offspring by mutation and by recombination. (Self-)adaptation can help in finding reasonable or sometimes even optimum parameter settings. Preliminary experiments have been conducted with an adaptive variant of the MA outlined in chapter 3 with first promising results. Extensive studies have to be made to show under which conditions the adaptive framework is capable of finding the globally optimum parameter settings.

An important aspect in memetic search has not been addressed in this work. Compared to other evolutionary approaches, the computation times required by the components of the algorithm have a great importance. The neighborhood search in a MA is a central component for which the number of iterations performed and thus the computation time required varies. Selection in the MA outlined in chapter 3 only takes the solution quality into account. However, not only the fitness of a solution is important but also the time required to find this solution. Especially in an adaptive framework, where search strategies compete with each other, it is essential to take the objective solution value *and* the running time of an 'agent' into account when selection is performed. Thus, it is of interest how much can be gained by replacing the static MA design used in this work with an adaptive asynchronous design.

For each combinatorial optimization problem investigated, there are issues for future research:

- *NK*-landscapes offer a simple model for investigating the dependencies of the components in the solution vector of binary coded problems. As the studies of the bipartitioning problems have shown, the structure of the dependency graph has strong influence on the problem characteristics. The influence is worth being studied in more detail in the *NK*-model.
- The studies on the unconstrained binary quadratic programming problem indicate that the existing problem instances are too easy for heuristic search. Large instances with higher landscape ruggedness should be generated to provide harder test cases. The fitness landscapes of combinatorial problems transformed into equivalent binary quadratic programs should be analyzed and compared with other BQP landscapes to gain more insight into the general landscape structure of the BQP.
- The experiments of memetic algorithms for large instances of the TSP have shown that parallel/distributed algorithms are required for solving instances with more than 10000 cities. Workstation clusters offer the ability to solve these instances in reasonable time. Thus, future research should focus on developing distributed memetic algorithms and distributed iterated local search heuristics for the TSP.
- The graph bipartitioning problem is a special case of the graph partitioning problem. The studies on the graph bipartitioning problem can be extended to the general case in future work.
- The results obtained from various experiments on the quadratic assignment problem raise the issue of self-adapting algorithms that are capable of choosing the best search strategy in memetic algorithms, i.e., variation by mutation or recombination, as well as the best parameter for the mutation operator (mutation jump distance). Initial experiments have shown that self-adaptation can aid in finding the most promising regions in the search space quickly, but to reveal the full potential of self-adapting memetic algorithms for the QAP, further detailed studies are required.

Most importantly, the research should be focused on searching for alternative ways to analyze fitness landscapes to identify other problem characteristics having influence on the effectiveness of heuristics. Therefore, a better understanding of the dynamics of the search process in a heuristic is required. As a consequence, more reliable tools for performance prediction and more effective heuristics will result.

Bibliography

[1] E. Aarts, P. van Laarhoven, J. Lenstra, and N. Ulder, “A Computational Study of Local Search Algorithms for Job–shop Scheduling,” *ORSA Journal on Computing*, vol. 6, pp. 118–125, 1994.

[2] E. H. L. Aarts and J. K. Lenstra, “Simulated Annealing,” in *Local Search in Combinatorial Optimization*, (E. H. L. Aarts and J. K. Lenstra, eds.), ch. 1, pp. 1–17, Wiley, 1997.

[3] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers, 1987.

[4] B. Alidaee, B. G. Kochenberger, and A. Ahmadian, “0–1 Quadratic Programming Approach for the Optimal Solution of Two Scheduling Problems,” *International Journal of Systems Science*, vol. 25, pp. 401–408, 1994.

[5] L. Altenberg, “Fitness Distance Correlation Analysis: An Instructive Counterexample,” in *Proceedings of the 7th International Conference on Genetic Algorithms*, (T. Bäck, ed.), pp. 57–64, Morgan Kaufmann, 1997.

[6] M. M. Amini, B. Alidaee, and G. A. Kochenberger, “A Scatter Search Approach to Unconstrained Quadratic Binary Programs,” in *New Ideas in Optimization*, (D. Corne, M. Dorigo, and F. Glover, eds.), pp. 317–329, London: McGraw-Hill, 1999.

[7] E. Angel and V. Zissimopoulos, “On the Hardness of the Quadratic Assignment Problem with Meta–Heuristics,” Tech. Rep., Laboratoire de Recherche en Informatique, Université Paris Sud, France, 1997.

[8] E. Angel and V. Zissimopoulos, “On the Landscape Ruggedness of the Quadratic Assignment Problem,” Tech. Rep., Laboratoire de Recherche en Informatique, Université Paris Sud, France, 1997.

[9] E. Angel and V. Zissimopoulos, “Autocorrelation Coefficient for the Graph Bipartitioning Problem,” *Theoretical Computer Science*, vol. 191, pp. 229–243, 1998.

[10] D. Applegate, R. Bixby, V. Chvátal, and B. Cook, “Finding Cuts in the TSP (A preliminary report),” Technical Report 95-05, DIMACS, 1995.

[11] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, “On the Solution of Traveling Salesman Problems,” *Documenta Mathematica*, vol. Extra Volume ICM III, pp. 645–656, 1998.

[12] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, "Concorde Benchmarks on TSPLIB Instances," W. M. Keck Center for Computational Discrete Optimization, Rice University, Houston, USA, 2000. http://www.keck.caam.rice.edu/concorde/bench.html.

[13] G. C. Armour and E. S. Buffa, "A Heuristic Algorithm and Simulation Approach to the Relative Location of Facilities," *Management Science*, vol. 9, no. 2, pp. 294–309, 1963.

[14] V. Bachelet, P. Preux, and E.-G. Talbi, "Parallel Hybrid Meta–Heuristics: Application to the Quadratic Assignment Problem," in *Proceedings of the Parallel Optimization Colloquium*, (Versailles, France), 1996.

[15] T. Bäck, U. Hammel, and H.-P. Schwefel, "Evolutionary Computation: Comments on the History and Current State," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 3–17, 1997.

[16] T. Bäck and H.-P. Schwefel, "An Overview of Evolutionary Algorithms for Parameter Optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 1–24, 1993.

[17] J. E. Baker, "Adaptive Selection Methods for Genetic Algorithms," in *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pp. 101–111, Carnegie Mellon publishers, 1985.

[18] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming – An Introduction.* Morgan Kaufmann, 1998.

[19] F. Barahona, M. Jünger, and G. Reinelt, "Experiments in Quadratic 0-1 Programming," *Mathematical Programming*, vol. 44, pp. 127–137, 1989.

[20] L. Barnett, *Evolutionary Dynamics on Fitness Landscapes with Neutrality.* Master's thesis, School of Cognitive Sciences, University of East Sussex, UK, 1997.

[21] R. Battiti and A. Bertossi, "Differential Greedy for the 0–1 Equicut Problem," in *Proceedings of the DIMACS Workshop on Network Design: Connectivity and Facilities Location*, (D. Du and P. Pardalos, eds.), pp. 3–21, American Mathematical Society, 1998.

[22] R. Battiti and A. Bertossi, "Greedy, Prohibition, and Reactive Heuristics for Graph-Partitioning," *IEEE Transactions on Computers*, vol. 48, no. 4, pp. 361–385, 1999.

[23] R. Battiti and G. Tecchiolli, "The Reactive Tabu Search," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 126–140, 1994.

[24] E. B. Baum, "Towards Practical "Neural" Computation for Combinatorial Optimization Problems," in *Neural Networks for Computing*, (J. S. Denker, ed.), (Snowbird 1986), pp. 53–58, American Institute of Physics, New York, 1986.

[25] J. E. Beasley, "OR-Library: Distributing Test Problems by Electronic Mail," *Journal of the Operational Research Society*, vol. 41, no. 11, pp. 1069–1072, 1990.

[26] J. E. Beasley, "Heuristic Algorithms for the Unconstrained Binary Quadratic Programming Problem," Tech. Rep., Management School, Imperial College, London, UK, 1998.

[27] J. L. Bentley, "Experiments on Traveling Salesman Heuristics," in *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 91–99, 1990.

[28] J. L. Bentley, "K-d-Trees for Semidynamic Point Sets," in *Proceedings of the Sixth Annual ACM Symposium on Computational Geometry*, pp. 187–197, 1990.

[29] J. L. Bentley, "Fast Algorithms for Geometric Traveling Salesman Problems," *ORSA Journal on Computing*, vol. 4, no. 4, pp. 387–411, 1992.

[30] R. Berretta and P. Moscato, "The Number Partitioning Problem: An Open Challenge for Evolutionary Computation?," in *New Ideas in Optimization*, (D. Corne, M. Dorigo, and F. Glover, eds.), ch. 17, pp. 261–278, McGraw-Hill, London, 1999.

[31] H. Bersini, M. Dorigo, S. Langerman, G. Seront, and L. Gambardella, "Results of the First International Contest on Evolutionary Optimisation (1st ICEO)," in *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, (Nagoya, Japan), pp. 611–615, 1996.

[32] A. Billionnet and A. Sutter, "Minimization of a Quadratic Pseudo–Boolean Function," *European Journal of Operational Research*, vol. 78, pp. 106–115, 1994.

[33] R. E. Bland and D. F. Shallcross, "Large Traveling Salesman Problems Arising from Experiments in X–ray Crystallography: A Preliminary Report on Computation," *Operations Research Letters*, vol. 8, pp. 125–128, 1989.

[34] K. D. Boese, *Models for Iterative Global Optimization.* PhD thesis, University of Carlifornia, Los Angeles, USA, 1996.

[35] K. Boese, "Cost versus Distance in the Traveling Salesman Problem," Tech. Rep. TR-950018, UCLA CS Department, 1995.

[36] K. Boese, A. Kahng, and S. Muddu, "A New Adaptive Multi-start Technique for Combinatorial Global Optimizations," *Operations Research Letters*, vol. 16, pp. 101–113, 1994.

[37] L. B. Booker, D. E. Goldberg, and J. H. Holland, "Classifier Systems and Genetic Algorithms," *Artificial Intelligence*, vol. 40, pp. 235–282, 1989.

[38] G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control.* Holden Day, 1970.

[39] R. M. Brady, "Optimization Strategies Gleaned from Biological Evolution," *Nature*, vol. 317, pp. 804–806, 1985.

[40] H. Braun, "On Solving Traveling Salesman Problems by Genetic Algorithms," in *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, (H.-P. Schwefel and R. Männer, eds.), (Dortmund, Germany), pp. 129–133, Springer-Verlag, Berlin, Germany, 1-3 Oct. 1991.

[41] E. D. Brown, L. C. Huntley, and R. A. Spillance, “A Parallel Genetic Heuristic for the Quadratic Assignment Problem,” in *Proceedings of the Third Conference on Genetic Algorithms*, (J. D. Schaffer, ed.), pp. 406–415, Morgan Kaufmann, 1989.

[42] L. Brunetta, M. Conforti, and G. Rinaldi, “A Branch-and-Cut Algorithm for the Equicut Problem,” *Mathematical Programming*, vol. 78, no. 2, pp. 243–263, 1997.

[43] E. S. Buffa, G. C. Armour, and T. E. Vollmann, “Allocating Facilities with CRAFT,” *Harvard Business Review*, pp. 136–158, March 1964.

[44] T. G. Bui and B. R. Moon, “A New Genetic Approach for the Traveling Salesman Problem,” in *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 7–12, 1994.

[45] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser, “Graph Bisection Algorithms with Good Average Case Behavior,” *Combinatorica*, vol. 7, no. 2, pp. 171–191, 1987.

[46] T. N. Bui and B. R. Moon, “A Genetic Algorithm for a Special Class of the Quadratic Assignment Problem,” in *Quadratic Assignment and Related Problems*, (P. M. Pardalos and H. Wolkowicz, eds.), pp. 137–187, American Mathematical Society, 1994.

[47] T. N. Bui and B. R. Moon, “Genetic Algorithm and Graph Partitioning,” *IEEE Transactions on Computers*, vol. 45, no. 7, pp. 841–855, 1996.

[48] R. E. Burkard and U. Fincke, “Probabilistic Asymptotic Properties of Some Combinatorial Optimization Problems,” *Discrete Applied Mathematics*, vol. 12, pp. 21–29, 1985.

[49] R. E. Burkard, S. Karisch, and F. Rendl, “QAPLIB – A Quadratic Assignment Problem Library,” *European Journal of Operational Research*, vol. 55, pp. 115–119, 1991. Updated Version: `http://www.imm.dtu.dk/~sk/qaplib`.

[50] R. E. Burkard and F. Rendl, “A Thermodynamically Motivated Simulation Procedure for Combinatorial Optimization Problems,” *European Journal of Operational Research*, vol. 17, pp. 169–174, 1984.

[51] R. E. Burkhard and J. Offerman, “Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme,” *Zeitschrift für Operations Research*, vol. 21, pp. B121–B132, 1977.

[52] P. Cejchan, “Personal communication,” 1998.

[53] N. Christofides, “The Traveling Salesman Problem,” in *Combinatorial Optimization*, (N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, eds.), pp. 131–149, Wiley and Sons, 1979.

[54] G. Clarke and J. W. Wright, “Scheduling of Vehicles from a Central Depot to a Number of Delivery Points,” *Operations Research*, vol. 12, pp. 568–581, 1964.

[55] B. Codenotti, G. Manzini, L. Margara, and G. Resta, “Perturbation: An Efficient Technique for the Solution of Very Large Instances of the Euclidean TSP,” Tech. Rep. TR-93-035, International Computer Science Institute, Berkeley, CA, 1993.

[56] D. T. Connolly, "An Improved Annealing Scheme for the Quadratic Assignment Problem," *European Journal of Operational Research*, vol. 46, pp. 93–100, 1990.

[57] D. Corne, M. Dorigo, and F. Glover, eds., *New Ideas in Optimization.* McGraw-Hill, London, 1999.

[58] D. Corne and P. Ross, "Practical Issues and Recent Advances in Job- and Open-Shop Scheduling," in *Evolutionary Algorithms in Engineering Applications*, (D. Dasgupta and Z. Michalewicz, eds.), pp. 531–546, Springer, 1997.

[59] G. A. Croes, "A Method for Solving Traveling Salesman Problems," *Operations Research*, vol. 5, pp. 791–812, 1958.

[60] H. Crowder and M. W. Padberg, "Solving Large–Scale Symmetric Traveling Salesman Problems to Optimality," *Management Science*, vol. 26, pp. 495–509, 1980.

[61] V.-D. Cung, T. Mautor, P. Michelon, and A. Tavares, "A Scatter Search Based Approach for the Quadratic Assignment Problem," in *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC)*, (T. Baeck, Z. Michalewicz, and X. Yao, eds.), (Indianapolis, USA), pp. 165–170, IEEE Press, 1997.

[62] G. B. Dantzig, "Programming of Interdependent Activities," *Econometrica*, vol. 17, pp. 193–211, 1949.

[63] G. B. Dantzig, *Linear Programming and Extensions.* Princeton Univ. Press, 1963.

[64] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson, "Solution of a Large-Scale Traveling Salesman Problem," *Operations Research*, vol. 2, pp. 393–410, 1954.

[65] C. Darwin, *On the Origin of Species.* London: John Murray, 1859.

[66] D. Dasgupta, "Information Processing in the Immune System," in *New Ideas in Optimization*, (D. Corne, M. Dorigo, and F. Glover, eds.), ch. 10, pp. 161–165, McGraw-Hill, London, 1999.

[67] Y. Davidor, "Epistasis Variance: Suitability of a Representation to Genetic Algorithms," *Complex Systems*, vol. 4, no. 4, pp. 369–383, 1990.

[68] Y. Davidor, "Epistasis Variance: A Viewpoint on GA-Hardness," in *Foundations of Genetic Algorithms*, (G. J. E. Rawlins, ed.), pp. 23–35, Morgan Kaufmann, 1991.

[69] L. Davis, "Applying Adaptive Algorithms to Epistatic Domains," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1985.

[70] L. Davis, *Genetic Algorithms and Simulated Annealing.* London: Pitman, 1987.

[71] R. Dawkins, *The Selfish Gene.* Oxford University Press, 1976.

[72] K. A. De Jong, *Analysis and Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, 1975.

[73] J. W. Dicky and J. W. Hopkins, "Campus Building Arrangement Using TOPAZ," *Transportation Research*, vol. 6, pp. 59–68, 1972.

[74] W. Domschke, *Einführung in Operations-Research (Introduction to Operations Research)*. Springer, 1998.

[75] M. Dorigo and G. Di Caro, "The Ant Colony Optimization Meta-Heuristic," in *New Ideas in Optimization*, (D. Corne, M. Dorigo, and F. Glover, eds.), pp. 11–32, McGraw–Hill, 1999.

[76] M. Dorigo, V. Maniezzo, and A. Colorni, "Positive Feedback as a Search Strategy," Tech. Rep. 91–016, Politecnico di Milano, Milano, Italy, 1991.

[77] M. Dorigo, V. Maniezzo, and A. Colorni, "The Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, vol. 26, no. 1, pp. 29–41, 1996.

[78] M. Dorigo and L. M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[79] S. Droste, T. Jansen, and I. Wegener, "Perhaps Not a Free luch But At Least a Free Appetizer," in *GECCO-1999: Proceedings of the Genetic and Evolutionary Computation Conference*, (W. B. et al., ed.), pp. 833–839, Morgan Kauffman, 1999.

[80] G. Dueck and T. Scheuer, "Threshold Accepting: A General Purpose Optimization Algorithm Superior to Simulated Annealing," *Journal of Computational Physics*, vol. 90, pp. 161–175, 1990.

[81] R. Durbin, R. Szeliski, and A. Yuille, "An Analysis of the Elastic Net Approach to the Traveling Salesman Problem," *Neural Computation*, vol. 1, pp. 348–358, 1989.

[82] R. Durbin and D. Willshaw, "An Analogue Approach to the Travelling Salesman Problem Using an Elastic Net Method," *Nature*, vol. 326, pp. 689–691, 1987.

[83] J. Dzubera and D. Whitley, "Advanced Correlation Analysis of Operators for the Traveling Salesman Problem," in *Parallel Problem Solving from Nature - Proceedings of the third Workshop, PPSN III*, (H.-P. Schwefel and R. Männer, eds.), (Dortmund, Germany), pp. 68–77, Springer-Verlag, Berlin, Germany, 1994.

[84] M. Eigen, J. McCaskill, and P. Schuster, "The Molecular Quasispecies," *Advanced Chemical Physics*, vol. 75, pp. 149–263, 1989.

[85] A. N. Elshafei, "Hospital Layout as a Quadratic Assignment Problem," *Operations Research Quarterly*, vol. 28, pp. 167–179, 1977.

[86] L. J. Eshelman and J. D. Schaffer, "Preventing Premature Convergence in Genetic Algorithms by Preventing Incest," in *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 115–122, Morgan Kaufmann, 1991.

[87] L. Eshelman, K. Mathias, and J. D. Schaffer, "Convergence Controlled Variation," in *Foundations of Genetic Algorithms 4*, (R. K. Belew and M. D. Vose, eds.), pp. 203–224, Morgan Kaufman, 1997.

[88] L. Eshelman, "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination," in *Foundations of Genetic Algorithms*, (G. J. E. Rawlings, ed.), pp. 265–283, Morgan Kaufmann, 1991.

[89] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," in *Proceedings of the 19th ACM/IEEE Design Automation Conference (DAC'82)*, pp. 175–181, 1982.

[90] C.-N. Fiechter, "A Parallel Tabu Search Algorithm for Large Traveling Salesman Problems," *Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, vol. 51, pp. 243–267, 1994.

[91] M. Fischetti, J. J. S. González, and P. Toth, "A Branch-and-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem," *Operations Research*, vol. 45, pp. 378–394, 1997.

[92] M. Fischetti and P. Toth, "An Additive Bounding Procedure for the Asymmetric Travelling Salesman Problem," *Mathematical Programming*, vol. 53, pp. 173–197, 1992.

[93] C. Fleurent and J. A. Ferland, "Genetic Hybrids for the Quadratic Assignment Problem," in *Quadratic Assignment and Related Problems*, (P. M. Pardalos and H. Wolkowicz, eds.), pp. 137–187, Amer. Math. Soc., 1994.

[94] M. M. Flood, "The Traveling–Salesman Problem," *Operations Research*, vol. 4, pp. 61–75, 1956.

[95] D. B. Fogel, "Applying Evolutionary Programming to Selected Traveling Salesman Problems," *Cybernetics and Systems*, vol. 24, pp. 27–36, 1993.

[96] D. B. Fogel, "An Evolutionary Approach to the Traveling Salesman Problem," *Biological Cybernetics*, vol. 60, pp. 139–144, 1988.

[97] L. J. Fogel, A. J. Owens, and M. J. Walsh, "Artificial Intelligence through a Simulation of Evolution," in *Biophysics and Cybernetic Systems*, (M. Maxfield, A. Callahan, and L. J. Fogel, eds.), pp. 131–155, London: Macmillan & Co, 1965.

[98] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*. New York: John Wiley & Sons, 1966.

[99] C. M. Fonseca and P. J. Fleming, "Multiobjective genetic algorithms," in *IEE Colloquium on Genetic Algorithms for Control Systems Engineering*, pp. 6/1–6/5, 1993.

[100] C. M. Fonseca and P. J. Fleming, "On the Performance Assessment and Comparison of Stochastic Multiobjective Optimizers," in *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature - PPSN IV*, (H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), pp. 584–593, Springer, 1996.

[101] W. Fontana, P. F. Stadler, E. G. Bornberg-Bauer, T. Griesmacher, I. L. Hofacker, M. Tacker, P. Tarazona, E. D. Weinberger, and P. Schuster, "RNA Folding Landscapes and Combinatory Landscapes," *Physcal Review E*, vol. 47, pp. 2083–2099, 1993.

[102] L. R. Ford and D. R. Fulkerson, *Flows in Networks.* Princeton, NJ: Princeton Univ. Press, 1963.

[103] B. R. Fox and M. B. McMahon, "Genetic Operators for Sequencing Problems," in *Foundations of Genetic Algorithms*, (G. J. E. Rawlings, ed.), pp. 284–300, Morgan Kaufmann, 1991.

[104] M. L. Fredman, D. S. Johnson, L. A. McGeoch, and G. Ostheimer, "Data Structures for Traveling Salesmen," *Journal of Algorithms*, vol. 18, pp. 432–479, 1995.

[105] B. Freisleben and P. Merz, "A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems," in *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, (T. Bäck, H. Kitano, and Z. Michalewicz, eds.), pp. 616–621, IEEE Press, 1996.

[106] B. Freisleben and P. Merz, "New Genetic Local Search Operators for the Traveling Salesman Problem," in *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature - PPSN IV*, (H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), pp. 890–900, Springer, 1996.

[107] B. Freisleben, "Meta-Evolutionary Approaches," in *Handbook of Evolutionary Computation*, (T. Bäck, D. B. Fogel, and Z. Michalewicz, eds.), pp. C2.8.1–C2.8.12, Oxford University Press, 1997.

[108] R. M. French and A. Messinger, "Genes, Phenes and the Baldwin Effect: Learning and Evolution in a Simulated Population," in *Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems $ArtificialLifeIV$*, (R. A. Brooks and P. Maes, eds.), pp. 277–282, MIT Press, 1994.

[109] G. Gallo, P. L. Hammer, and B. Simeone, "Quadratic Knapsack Problems," *Mathematical Programming*, vol. 12, pp. 132–149, 1980.

[110] L. M. Gambardella, E. Taillard, and G. Agazzi, "MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows," Technical Report IDSIA-06-99, IDSIA - Instituto Dalle Molle di Studi sull'Intelligenza Artificiale, Lugano, Switzerland, 1999.

[111] L. Gambardella, É. Taillard, and M. Dorigo, "Ant Colonies for the QAP," *Journal of the Operations Research Society*, vol. 50, pp. 167–176, 1999.

[112] L. M. Gambardella and M. Dorigo, "Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem," in *Proc. 12th International Conference on Machine Learning*, pp. 252–260, Morgan Kaufmann, 1995.

[113] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[114] J. S. Gero, V. A. Kazakov, and T. Schnier, "Genetic Engineering and Design Problems," in *Evolutionary Algorithms in Engineering Applications*, (D. Dasgupta and Z. Michalewicz, eds.), pp. 47–68, Springer, 1997.

[115] F. Glover, G. Kochenberger, B. Alidaee, and M. Amini, "Tabu Search with Critical Event Memory: An Enhanced Application for Binary Quadratic Programs," in *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization*, (S. Voss, S. Martello, I. Osman, and C. Roucairol, eds.), pp. 83–109, Kluwer Academic Publishers, 1998.

[116] F. Glover, G. A. Kochenberger, and B. Alidaee, "Adaptive Memory Tabu Search for Binary Quadratic Programs," *Management Science*, vol. 44, no. 3, pp. 336–345, 1998.

[117] F. Glover and M. Laguna, *Tabu Search.* Kluwer Academic Publishers, 1998.

[118] D. E. Goldberg and J. R. Lingle, "Alleles, Loci, and the Traveling Salesman Problem," in *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pp. 154–159, Carnegie Mellon publishers, 1985.

[119] D. E. Goldberg, "Simple Genetic Algorithms and the Minimal, Deceptive Problem," in *Genetic Algorithms and Simulated Annealing*, pp. 74–88, Morgan Kaufmann, 1987.

[120] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, 1989.

[121] M. Gorges-Schleuter, "ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy," in *Proceedings of the Third International Conference on Genetic Algorithms*, (J. D. Schaffer, ed.), pp. 422–427, Morgan Kaufmann, 1989.

[122] M. Gorges-Schleuter, *Genetic Algorithms and Population Structures – A Massively Parallel Algorithm.* PhD thesis, Universität Dortmund, Germany, 1990.

[123] M. Gorges-Schleuter, "Asparagos96 and the Traveling Salesman Problem," in *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, pp. 171–174, IEEE Press, 1997.

[124] J. Grefenstette, R. Gopal, B. Rosimaita, and D. V. Gucht, "Genetic Algorithms for the Traveling Salesman Problem," in *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pp. 160–168, Carnegie Mellon publishers, 1985.

[125] J. J. Grefenstette, "Incooperating Problem Specific Knowledge into Genetic Algorithms," in *Genetic Algorithms and Simulated Annealing*, (L. Davis, ed.), pp. 42–60, Morgan Kaufmann Publishers, 1987.

[126] M. Grötschel, "On the Symmetric Traveling Salesman Problem: Solution of a 120-city problem," *Mathematical Programming Studies*, vol. 12, pp. 61–77, 1980.

[127] M. Grötschel and O. Holland, "Solution of Large-scale Symmetric Travelling Salesman Problems," *Mathematical Programming*, vol. 51, pp. 141–202, 1991.

[128] M. Grötschel, M. Jünger, and G. Reinelt, "Optimal Control of Plotting and Drilling Machines: a Case Study," *ZOR - Methods and Models of Operations Research*, vol. 35, pp. 61–84, 1991.

[129] M. Grötschel and M. W. Padberg, "Polyhedral Theory," in *The Traveling Salesman Problem*, (E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds.), ch. 8, pp. 251–305, John Wiley & Sons, 1985.

[130] J. Gu and X. Huang, "Efficient Local Search With Search Space Smoothing: A Case Study of the Traveling Salesman Problem (TSP)," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, pp. 728–735, 1994.

[131] R. W. Hamming, "Error Detecting and Error Correcting Codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[132] P. J. B. Hancock, "An Empirical Comparison of Selection Methods in Evolutionary Algorithms," in *Proceedings of the AISB Workshop on Evolutionary Computing*, (T. C. Fogarty, ed.), pp. 80–94, Springer, 1994.

[133] C. Helmberg and F. Rendl, "Solving Quadratic (0,1)-Problems by Semidefinite Programs and Cutting Planes," *Mathematical Programming*, vol. 82, pp. 291–315, 1998.

[134] B. Hendrickson and R. Leland, "A Multi-Level Algorithm For Partitioning Graphs," in *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, (S. Karin, ed.), ACM Press and IEEE Computer Society Press, 1995.

[135] B. Hendrickson and R. Leland, "An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations," *SIAM Journal on Scientific Computing*, vol. 16, no. 2, pp. 452–469, 1995.

[136] M. Herdy, "Application of the Evolutionsstrategie to Discrete Optimization Problems," in *Parallel Problem Solving from Nature*, (H.-P. Schwefel and R. Männer, eds.), pp. 188–192, Springer, 1991.

[137] J. Hertz, A. Krogh, and R. G. Palmer, *An Introduction to the Theory of Neural Computation.* Addison Wesley, 1991.

[138] W. D. Hillis, "Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure," in *Artificial life II*, (C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, eds.), pp. 313–324, Addison-Wesley, 1992.

[139] F. Hoffmeister and T. Bäck, "Genetic Algorithms and Evolution Strategies: Similarities and Differences," in *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, (H. P. Schwefel and R. Männer, eds.), pp. 455–469, Springer, 1991.

[140] F. Hoffmeister and T. Bäck, "Genetic Algorithms and Evolution Strategies: Similarities and Differences," Tech. Rep. SYS-1/92, University of Dortmund - Systems Analysis Research Group, 1992.

[141] C. Höhn and C. Reeves, "Graph Partitioning Using Genetic Algorithms," in *Proceedings of the 2nd International Conference on Massively Parallel Computing Systems*, (G. R. Sechi, ed.), pp. 27–43, IEEE Computer Society Press, 1996.

[142] J. Holland, *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[143] A. Homaifar, S. Guan, and G. E. Liepins, “A New Approach to the Traveling Salesman Problem by Genetic Algorithms,” in *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 460–466, Morgan Kaufmann, 1993.

[144] J. N. Hooker, “Testing Heuristics: We Have It All Wrong,” *Heuristics*, vol. 1, pp. 33–42, 1996.

[145] J. J. Hopfield and D. W. Tank, “‘Neural’ Computation of Decisions in Optimization Problems,” *Biological Cybernetics*, vol. 52, pp. 141–152, 1985.

[146] W. Horjik, *Population Flow on Fitness Landscapes.* Master’s thesis, Erasmus University Rotterdam, Department of Computer Science, 1994.

[147] W. Horjik and B. Manderick, “The Usefulness of Recombination,” in *Proceedings of the European Conference on Artificial Life*, p. , Springer, 1995.

[148] J. Horn and D. E. Goldberg, “Genetic Algorithm Difficulty and the Modality of Fitness Landscapes,” in *Proceedings of the Third Workshop on Foundations of Genetic Algorithms*, (L. D. Whitley and M. D. Vose, eds.), pp. 243–270, Morgan Kaufmann, 1995.

[149] H. Inayoshi and B. Manderick, “The Weighted Graph Bi-Partitioning Problem: A Look at GA Performance,” in *Parallel Problem Solving From Nature – PPSN III*, (Y. Davidor and H.-P. Schwefel, eds.), pp. 617–625, Springer, 1994.

[150] S. Ishii and M. Sato, “Constrained Neural Approaches to Quadratic Assignment Problems,” *Neural Networks*, vol. 11, pp. 1073–1082, 1998.

[151] P. L. Ivănescu, “Some Network Flow Problems Solved with Pseudo-Boolean Programming,” *Operations Research*, vol. 13, pp. 388–399, 1965.

[152] F. T. J. Carrizo and P. Moscato, “A Computational Ecology for the Quadratic Assignment Problem,” in *Proceedings of the 21st Meeting on Informatics and Operations Research*, (Buenos Aires), SADIO, 1992.

[153] C.-S. Jeong and M.-H. Kim, “Fast Parallel Simulated Annealing for Traveling Salesman Problem on SIMD Machines with Linear Interconnections,” *Parallel Computing*, vol. 17, no. 2–3, pp. 221–228, 1991.

[154] P. Jog, J. Y. Suh, and D. V. Gucht, “The Effects of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Travelling Salesman Problem,” in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 110–115, Morgan Kaufman, 1989.

[155] P. Jog, J. Y. Suh, and D. V. Gucht, “Parallel Genetic Algorithms Applied to the Traveling Salesman Problem,” *SIAM Journal on Optimization*, vol. 1, no. 4, pp. 515–529, 1991.

[156] D. S. Johnson, “Local Optimization and the Traveling Salesman Problem,” in *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, pp. 446–461, Springer, Berlin, 1990.

[157] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by Simulated Annealing; Part I, Graph Partitioning," *Operations Research*, vol. 37, pp. 865–892, 1989.

[158] D. S. Johnson and L. A. McGeoch, "The Traveling Salesman Problem: A Case Study," in *Local Search in Combinatorial Optimization*, (E. H. L. Aarts and J. K. Lenstra, eds.), pp. 215–310, Wiley and Sons, New York, 1997.

[159] D. S. Johnson and C. H. Papadimitriou, "Computational Complexity," in *The Traveling Salesman Problem*, (E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds.), ch. 3, pp. 37–85, John Wiley & Sons, 1985.

[160] T. Jones and S. Forrest, "Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms," in *Proceedings of the 6th International Conference on Genetic Algorithms*, (L. J. Eshelman, ed.), pp. 184–192, Morgan Kaufmann, 1995.

[161] M. Jünger and G. Rinaldi, "Relaxations of the Max Cut Problem and Computation of Spin Glass Ground States," in *Proceedings of the Symposium on Operations Research (SOR'97)*, (P. Kischka, ed.), pp. 74–83, Springer, 1998.

[162] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Artificial Inteligence Research*, vol. 4, pp. 237–285, 1996.

[163] G. Karypis and V. Kumar, "Multilevel Graph Partitioning Schemes," in *Proceedings of the 24th International Conference on Parallel Processing*, (Oconomowoc, WI), pp. 113–122, 1995.

[164] K. Katayama, "Personal Communication," 2000.

[165] K. Katayama and H. Narihisa, "Iterated Local Search Approach using Genetic Transformation to the Traveling Salesman Problem," in *GECCO-1999: Proceedings of the Genetic and Evolutionary Computation Conference*, (W. B. et al., ed.), pp. 321–328, Morgan Kauffman, 1999.

[166] K. Katayama and H. Narihisa, "Performance of Simulated Annealing-based Heuristic for the Unconstrained Binary Quadratic Programming Problem," Tech. Rep., Okayama University of Science, Dept. of Information and Computer Engineering, Okayama, Japan, 1999.

[167] K. Katayama and H. Narihisa, "Solving Large Binary Quadratic Programming Problems by Effective Genetic Local Search Algorithm," in *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kauffman, 2000. to appear.

[168] S. A. Kauffman, *The Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, 1993.

[169] S. A. Kauffman and S. Levin, "Towards a General Theory of Adaptive Walks on Rugged Landscapes," *Journal of Theoretical Biology*, vol. 128, pp. 11–45, 1987.

[170] J. Kennedy and R. C. Eberhart, “The Particle Swarm: Social Adaptation in Information-Processing Systems,” in *New Ideas in Optimization*, (D. Corne, M. Dorigo, and F. Glover, eds.), ch. 25, pp. 379–388, McGraw-Hill, London, 1999.

[171] B. Kernighan and S. Lin, “An Efficient Heuristic Procedure for Partitioning Graphs,” *Bell Systems Journal*, vol. 49, pp. 291–307, 1972.

[172] M. Kimura, *The Neutral Theory of Molecular Evolution.* Cambridge University Press, 1883.

[173] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by Simulated Annealing,” *Science*, vol. 220, pp. 671–680, 1983.

[174] T. Kohonen, *Self–Organisation and Associative Memory.* Springer, 3rd edition, 1990.

[175] A. Kolen and E. Pesch, “Genetic Local Search in Combinatorial Optimization,” *Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, vol. 48, pp. 273–284, 1994.

[176] T. C. Koopmans and M. J. Beckmann, “Assignment Problems and the Location of Economic Activities,” *Econometrica*, vol. 25, pp. 53–76, 1957.

[177] B. Korte, “Applications of Combinatorial Optimization,” in *Talk at the 13th International Mathematical Programming Symposium*, (Tokyo), 1988.

[178] B. Korte, “Applications of Combinatorial Optimization,” in *Mathematical Programming: Recent Development and Applications*, (M. Iri and K. Tanabe, eds.), pp. 203–225, Kluwer Academic Publishers, 1989.

[179] J. R. Koza, *Genetic Programming: On the Programming of Computers by Natural Selection.* Cambridge, MA: MIT Press, 1992.

[180] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge: MIT Press, 1994.

[181] B. Krakhofer and P. F. Stadler, “Local Minima in the Graph Bipartitioning Problem,” *Europhys. Lett.*, vol. 34, pp. 85–90, 1996.

[182] J. Krarup and P. M. Pruzan, “Computer-Aided Layout Design,” *Mathematical Programming Study*, vol. 9, pp. 75–94, 1978.

[183] J. B. Kruskal, “On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem,” *Proceedings of the American Mathematical Society*, vol. 7, pp. 48–50, 1956.

[184] G. Laszewski and H. Mühlenbein, “Partitioning a Graph with a Parallel Genetic Algorithm,” in *Parallel Problem Solving from Nature*, (H.-P. Schwefel and R. Männer, eds.), (Berlin), pp. 165–169, Springer, 1991.

[185] D. J. Laughunn, “Quadratic Binary Programming,” *Operations Research*, vol. 14, pp. 454–461, 1970.

[186] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* New York: Wiley and Sons, 1985.

[187] J. K. Lenstra and A. H. G. R. Kan, "Some Simple Applications of the Travelling Salesman Problem," *Opl Res. Q.*, vol. 26, pp. 717–733, 1975.

[188] Y. Li and P. M. Pardalos, "Generating Quadratic Assignment Test Problems with Known Optimal Permutations," *Computational Optimization and Applications*, vol. 1, pp. 163–184, 1992.

[189] Y. Li, P. M. Pardalos, and M. G. C. Resendre, "A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem," in *Quadratic Assignment and Related Problems*, (P. M. Pardalos and H. Wolkowicz, eds.), pp. 237–261, Amer. Math. Soc., 1994.

[190] G. E. Liepins and M. R. Hilliard, "Greedy Genetics," in *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pp. 90–99, Lawrence Erlbaum, 1987.

[191] S. Lin, "Computer Solutions of the Travelling Salesman Problem," *Bell System Tech. Journal*, vol. 44, pp. 2245–2269, 1965.

[192] S. Lin and B. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research*, vol. 21, pp. 498–516, 1973.

[193] J. D. Litke, "An Improved Solution to the Traveling Salesman Problem with Thousands of Nodes," *Communications of the ACM*, vol. 27, no. 12, pp. 1227–1236, 1984.

[194] A. Lodi, K. Allemand, and T. M. Liebling, "An Evolutionary Heuristic for Quadratic 0–1 Programming," *European Journal of Operational Research*, vol. 119, pp. 662–670, 1999.

[195] A. Lucena and J. E. Beasley, "A Branch and Cut Algorithm for the Steiner Problem in Graphs," *Networks: An International Journal*, vol. 31, pp. 39–59, 1998.

[196] K.-T. Mak and A. J. Morton, "Distances between Traveling Salesman Tours," *Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, vol. 58, pp. 281–291, 1995.

[197] B. Manderick, M. de Weger, and P. Spiessens, "The Genetic Algorithm and the Structure of the Fitness Landscape," in *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 143–150, Morgan Kaufmann, 1991.

[198] V. Maniezzo, "Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem," Tech. Rep. CSR 98-1, C.L. in Scienze dell'Informazione, Universitá di Bologna, Sede di Cesena, Italy, 1998.

[199] V. Maniezzo, A. Colorni, and M. Dorigo, "The Ant System Applied to the Quadratic Assignment Problem," Tech. Rep. 94/28, IRIDIA, Université de Bruxelles, 1994.

[200] V. Maniezzo, M. Dorigo, and A. Colorni, "Algodesk: An Experimental Comparison of Eight Evolutionary Heuristics Applied to the Quadratic Assignment Problem," *European Journal of Operational Research*, vol. 81, pp. 188–204, 1995.

[201] F. Margot, "Quick Updates for p-opt TSP Heuristics," *Operations Research Letters*, vol. 11, pp. 45–46, 1992.

[202] O. Martin, S. W. Otto, and E. Felten, "Large-Step Markov Chains for the TSP Incorporating Local Search Heuristics," *Operations Research Letters*, vol. 11, pp. 219–224, 1992.

[203] O. Martin, S. W. Otto, and E. W. Felten, "Large-Step Markov Chains for the Traveling Salesman Problem," *Complex Systems*, vol. 5, pp. 299–326, 1991.

[204] K. Mathias and D. Whitley, "Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem," in *Parallel Problem Solving from Nature - Proceedings of 2nd Workshop, PPSN 2*, (R. Männer and B. Manderick, eds.), pp. 219–228, Elsevier Science Publishers, 1992.

[205] R. D. McBride and J. S. Yormark, "An Implicit Enumeration Algorithm for Quadratic Integer Programming," *Management Science*, vol. 26, no. 3, pp. 282–296, 1980.

[206] J. McCarthy, "LISP: A Programming System for Symbolic Manipulations," Report, ACM Annual Meeting, Cambridge, Mass., 1959.

[207] E. J. McCormick, *Human Factors Engineering.* New York: McGraw-Hill, 1970.

[208] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Idea Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[209] P. Merz and B. Freisleben, "A Genetic Local Search Approach to the Quadratic Assignment Problem," in *Proceedings of the 7th International Conference on Genetic Algorithms*, (T. Bäck, ed.), pp. 465–472, Morgan Kaufmann, 1997.

[210] P. Merz and B. Freisleben, "Genetic Local Search for the TSP: New Results," in *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, (T. Bäck, Z. Michalewicz, and X. Yao, eds.), pp. 159–164, IEEE Press, 1997.

[211] P. Merz and B. Freisleben, "Memetic Algorithms and the Fitness Landscape of the Graph Bi-Partitioning Problem," in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature - PPSN V*, (A.-E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, eds.), pp. 765–774, Springer, 1998.

[212] P. Merz and B. Freisleben, "On the Effectiveness of Evolutionary Search in High–Dimensional NK-Landscapes," in *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, (D. Fogel, ed.), pp. 741–745, IEEE Press, 1998.

[213] P. Merz and B. Freisleben, "A Comparison of Memetic Algorithms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem," in *1999 Congress on Evolutionary Computation (CEC'99)*, (P. Angeline, ed.), pp. 2063–2070, IEEE Press, 1999.

[214] P. Merz and B. Freisleben, “Fitness Landscapes and Memetic Algorithm Design,” in *New Ideas in Optimization*, (D. Corne, M. Dorigo, and F. Glover, eds.), pp. 245–260, McGraw–Hill, 1999.

[215] P. Merz and B. Freisleben, “Genetic Algorithms for Binary Quadratic Programming,” in *GECCO-1999: Proceedings of the Genetic and Evolutionary Computation Conference*, (W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, eds.), pp. 417–424, Morgan Kauffman, 1999.

[216] P. Merz and B. Freisleben, “Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem,” *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 4, pp. 337–352, 2000.

[217] P. Merz and B. Freisleben, “Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning,” *Evolutionary Computation*, vol. 8, no. 1, pp. 61–91, 2000.

[218] P. Merz and B. Freisleben, “Greedy and Local Search Heuristics for Uncontrained Binary Quadratic Programming,” Tech. Rep., Department of Electrical Engineering and Computer Science, University of Siegen, Germany, 2000. Accepted for publication in *Journal of Heuristics*.

[219] P. Merz, *Genetische Algorithmen für kombinatorische Optimierungsprobleme*. Master’s thesis, University of Siegen, Germany, 1996.

[220] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, “Equation of State Calculation by Fast Computing Machines,” *Journal of Chemical Physics*, vol. 21, pp. 1087–1092, 1953.

[221] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer, 1996.

[222] D. L. Miller and J. F. Pekny, “Exact Solution of Large Asymmetric Traveling Salesman Problems,” *Science*, vol. 251, pp. 754–761, 1991.

[223] C. Ming and L. Minghui, “Kohonen neural network-based solution of TSP,” *Mini-Micro Systems*, vol. 15, no. 11, pp. 35–9, 1994.

[224] A. Möbius, A. Diaz-Sanchez, B. Freisleben, M. Schreiber, A. Fachat, K. Hoffmann, P. Merz, and A. Neklioudov, “Two Physically Motivated Algorithms for Combinatorial Optimization: Thermal Cycling and Iterative Partial Transcription,” *Computer Physics Communications*, vol. 121–122, no. 1–3, pp. 34–36, 1999.

[225] A. Möbius, B. Freisleben, P. Merz, and M. Schreiber, “Combinatorial Optimization by Iterative Partial Transcription,” *Physical Review E*, vol. 59, no. 4, pp. 4667–4674, 1999.

[226] P. Moscato, “On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms,” Tech. Rep. No. 790, Caltech Concurrent Computation Program, California Institue of Technology, 1989.

[227] P. Moscato and M. G. Norman, “A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems,” in *Parallel Computing and Transputer Applications*, (M. Valero, E. Onate, M. Jane, J. L. Larriba, and B. Suarez, eds.), (Amsterdam), pp. 177–186, IOS Press, 1992.

[228] P. Moscato and M. Norman, “Arbitrarily Large Planar ETSP Instances with Known Optimal Tours,” *Pesquisa Operacional*, vol. 15, pp. 89–96, 1995.

[229] P. Moscato and M. Norman, “On the performance of Heuristics on Finite and Infinite Fractal Instancesof the Euclidean Traveling Salesman Problem,” *INFORMS Journal on Computing*, vol. 10, no. 2, pp. 121–132, 1998.

[230] P. Moscato, “Memetic Algorithms: A Short Introduction,” in *New Ideas in Optimization*, (D. Corne, M. Dorigo, and F. Glover, eds.), ch. 14, pp. 219–234, McGraw-Hill, London, 1999.

[231] H. Mühlenbein, “Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization,” in *Proceedings of the Third International Conference on Genetic Algorithms*, (Schaffer, ed.), pp. 416–421, Morgan Kaufmann, 1989.

[232] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, “Evolution Algorithms in Combinatorial Optimization,” *Parallel Computing*, vol. 7, pp. 65–88, 1988.

[233] H. Mühlenbein, “Evolution in Time and Space – The Parallel Genetic Algorithm,” in *Foundations of Genetic Algorithms*, (G. J. E. Rawlins, ed.), Morgan Kaufmann Publishers, 1991.

[234] Y. Nagata and S. Kobayashi, “Edge Assembly Crossover: A High–power Genetic Algorithm for the Traveling Salesman Problem,” in *Proceedings of the 7th International Conference on Genetic Algorithms*, (T. Bäck, ed.), pp. 450–457, Morgan Kaufmann, 1997.

[235] D. Neto, *Efficient Cluster Compensation for Lin-Kernighan Heuristics.* PhD thesis, University of Toronto, Computer Science Department, 1999.

[236] V. Nissen and H. Paul, “A Modification of Threshold Accepting and its Application to the Quadratic Assignment Problem,” *OR Spektrum*, vol. 17, pp. 205–210, 1995.

[237] V. Nissen, “Solving the Quadratic Assignment Problem with Clues from Nature,” *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 66–72, 1994.

[238] M. G. Norman and P. Moscato, “The Euclidean Traveling Salesman Problem and a Space-Filling Curve,” *Chaos, Solitons and Fractals*, vol. 6, pp. 389–397, 1995.

[239] I. M. Oliver, D. J. Smith, and J. R. C. Holland, “A Study of Permutation Crossover Operators on the Traveling Salesman Problem,” in *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pp. 224–230, Lawrence Erlbaum, 1987.

[240] F. Oppacher and M. Wineberg, "The Shifting Balance Genetic Algorithm: Improving the GA in a Dynamic Environment," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, (W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, eds.), pp. 504–510, Morgan Kaufmann, 1999.

[241] I. Or, *Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking*. PhD thesis, Northwestern University, Evanston, 1976.

[242] M. Padberg and G. Rinaldi, "Optimization of a 532-city Symmetric Traveling Salesman Problem by Branch and Cut," *Operations Research Letters*, vol. 6, pp. 1–7, 1987.

[243] M. Padberg and G. Rinaldi, "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems," *SIAM Review*, vol. 33, pp. 60–100, March 1991.

[244] P. M. Pardalos and G. P. Rodgers, "Computational Aspects of a Branch and Bound Algorithm for Unconstrained Quadratic Zero–One Programming," *Computing*, vol. 45, pp. 131–144, 1990.

[245] P. M. Pardalos and G. P. Rodgers, "A Branch and Bound Algorithm for the Maximum Clique Problem," *Computers and Operations Research*, vol. 19, no. 5, pp. 363–375, 1992.

[246] P. M. Pardalos and J. Xue, "The Maximum Clique Problem," *Journal of Global Optimization*, vol. 4, pp. 301–328, 1994.

[247] J. Paredis, "Coevolutionary Computation," *Artificial Life*, vol. 2, no. 4, pp. 355–375, 1995.

[248] C. Peterson and B. Soderberg, "Artificial Neural Networks," in *Modern Heuristic Techniques for Combinatorial Problems*, (C. R. Reeves, ed.), McGraw Hill, 1993.

[249] C. Peterson and J. Anderson, "Neural Networks and NP-complete Optimization Problems; A Performance Study on the Graph Bisection Problem," *Complex Systems*, vol. 2, no. 1, pp. 59–89, 1988.

[250] A. T. Phillips and J. B. Rosen, "A Quadratic Assignment Formulation for the Molecular Conformation Problem," *Journal of Global Optimization*, vol. 4, pp. 229–241, 1994.

[251] R. D. Plante, T. J. Lowe, and R. Chandrasekaran, "The Product Matrix Traveling Salesman Problem: An Application and Solution Heuristics," *Operations Research*, vol. 35, pp. 772–783, 1987.

[252] A. Pothen, "Graph Partitioning Algorithms with Applications to Scientific Computing," Tech. Rep. TR–97–03, Old Dominion University, 1997.

[253] J.-Y. Potvin, "The Traveling Salesman Problem: A Neural Network Perspective," *ORSA Journal on Computing*, vol. 5, pp. 328–348, 1993.

[254] N. Radcliffe and P. Surry, "Fitness Variance of Formae and Performance Prediction," in *Proceedings of the Third Workshop on Foundations of Genetic Algorithms*, (L. Whitley and M. Vose, eds.), (San Francisco), pp. 51–72, Morgan Kaufmann, 1994.

[255] N. Radcliffe and P. Surry, "Formal Memetic Algorithms," in *Evolutionary Computing: AISB Workshop*, (T. Fogarty, ed.), pp. 1–16, Springer-Verlag, Berlin, 1994.

[256] S. Rana, L. D. Whitley, and R. Cogswell, "Searching in the Presence of Noise," in *Parallel Problem Solving from Nature - PPSN IV*, (H. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), (Berlin), pp. 198–207, Springer, 1996.

[257] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Stuttgart: Frommann-Holzboog, 1973.

[258] I. Rechenberg and P. Bienert, "Evolutionsstrategie," Tech. Rep., TU Berlin, 1969.

[259] C. R. Reeves, "Landscapes, Operators and Heuristic Search," Tech. Rep., School of Mathematical and Information Science, Coventry University, Coventry, UK, 1997. Accepted for publication in Annals of Operations Research.

[260] G. Reinelt, "TSPLIB— A Traveling Salesman Problem Library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[261] G. Reinelt, *The Traveling Salesman: Computational Solutions for TSP Applications.* Vol. 840 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, 1994.

[262] R. G. Reynolds, "Cultural Algorithms: Theory and Applications," in *New Ideas in Optimization*, (D. Corne, M. Dorigo, and F. Glover, eds.), ch. 24, pp. 367–377, McGraw-Hill, London, 1999.

[263] S. Ronald, "Finding Multible Solutions with an Evolutionary Algorithm," in *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation*, pp. 641–646, IEEE Press, 1995.

[264] S. Ronald, "Distance Functions for Order–Based Encodings," in *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, pp. 49–54, IEEE Press, 1997.

[265] B. Roy and B. Sussmann, "Les Problèmes D'Ordonnancement Avec Constraints Disjonctives," Note DS 9 bis, SEMA, Paris, France, 1974.

[266] G. Rudolph, "Global Optimization by Means of Distributed Evolution Strategies," in *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, (H. P. Schwefel and R. Männer, eds.), pp. 209–213, Springer, 1991.

[267] H. Salkin and K. Mathur, *Foundations of Integer Programming.* North Holland, 1989.

[268] P. Schuster, W. Fontana, P. F. Stadler, and I. L. Hofacker, "From Sequences to Shapes and Back: A Case Study in RNA Secondary Structures," *Proc. Roy. Soc. Lond. B*, vol. 255, pp. 279–284, 1994.

[269] H.-P. Schwefel, *Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie.* Vol. 26 of *Interdisciplinary Systems Research*, Basel: Birkhäuser Verlag, 1977.

[270] H.-P. Schwefel, *Numerical Optimization of Computer Models.* Wiley, 1981.

[271] H. D. Simon, "Partitioning of Unstructured Problems for Parallel Processing," *Computing Systems in Engineering*, vol. 2, no. 3, pp. 135–148, 1991.

[272] C. D. Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, and G. Rinaldi, "Exact Ground States in Spin Glasses: New Experimental Results with a Branch and Cut Algorithm," *Journal of Statistical Physics*, vol. 80, pp. 487–496, 1995.

[273] J. Skorin-Kapov, "Tabu Search Applied to the Quadratic Assignment Problem," *ORSA Journal on Computing*, vol. 2, no. 1, pp. 33–45, 1990.

[274] J. Skorin-Kapov, "Extensions of a Tabu Search Adaptation to the Quadratic Assignment Problem," *Computers and Operations Research*, vol. 21, no. 8, pp. 855–865, 1994.

[275] G. B. Sorkin, "Efficient Simulated Annealing on Fractal Energy Landscapes," *Algorithmica*, vol. 6, pp. 367–418, 1991.

[276] D. A. Spielman and S.-H. Teng, "Spectral Partitioning Works: Planar Graphs and Finite Element Meshes," in *37th Annual Symposium on Foundations of Computer Science*, (Burlington, Vermont), pp. 96–105, 1996.

[277] P. F. Stadler, "Correlation in Landscapes of Combinatorial Optimization Problems," *Europhys. Lett.*, vol. 20, pp. 479–482, 1992.

[278] P. F. Stadler, "Towards a Theory of Landscapes," in *Complex Systems and Binary Networks*, (R. Lopéz-Peña, R. Capovilla, R. García-Pelayo, H. Waelbroeck, and F. Zertuche, eds.), (Berlin, New York), pp. 77–163, Springer Verlag, 1995.

[279] P. F. Stadler, "Landscapes and their Correlation Functions," *J. Math. Chem.*, vol. 20, pp. 1–45, 1996.

[280] P. F. Stadler and R. Happel, "Correlation Structure of the Landscape of the Graph-Bipartitioning-Problem," *J. Phys. A:Math. Gen.*, vol. 25, pp. 3103–3110, 1992.

[281] P. F. Stadler and W. Schnabl, "The Landscape of the Travelling Salesman Problem," *Physics Letters A*, vol. 161, pp. 337–344, 1992.

[282] P. F. Stadler and G. P. Wagner, "The Algebraic Theory of Recombination Spaces," *Evolutionary Computation*, vol. 5, pp. 241–275, 1998.

[283] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, and C. Whitley, "A Comparison of Genetic Sequencing Operators," in *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 69–76, Morgan Kaufmann, 1991.

[284] A. G. Steenbeek, E. Marchiori, and A. E. Eiben, "Finding Balanced Graph Bi-Partitions Using a Hybrid Genetic Algorithm," in *Proceedings of the IEEE International Conference on Evolutionary Computation ICEC'98*, pp. 90–95, IEEE Press, 1998.

[285] L. Steinberg, “The Backboard Wiring Problem: A Placement Algorithm,” *SIAM Review*, vol. 3, pp. 37–50, 1961.

[286] T. Stützle, “$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System for Quadratic Assignment Problems,” Tech. Rep. AIDA–97–4, FG Intellektik, TU Darmstadt, Germany, 1997.

[287] T. Stützle, *Local Search Algorithms for Combinatorial Problems — Analysis, Improvements, and New Applications.* PhD thesis, FB Informatik, TU Darmstadt, 1998.

[288] T. Stützle and M. Dorigo, “ACO Algorithms for the Quadratic Assignment Problem,” in *New Ideas in Optimization*, (D. Corne, M. Dorigo, and F. Glover, eds.), pp. 33–50, McGraw–Hill, London, 1999.

[289] T. Stützle and H. Hoos, “The $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System and Local Search for the Traveling Salesman Problem,” in *Proceedings 1997 IEEE International Conference on Evolutionary Computation (ICEC’97)*, (T. Baeck, Z. Michalewicz, and X. Yao, eds.), pp. 309–314, 1997.

[290] T. Stützle and H. Hoos, “$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System and Local Search for Combinatorial Optimization Problems,” in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, (S. Voss, S. Martello, I. Osman, and C. Roucairol, eds.), pp. 313–329, Kluwer, Boston, 1999.

[291] J. Y. Suh and D. V. Gucht, “Incorporating Heuristic Information into Genetic Search,” in *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pp. 100–107, Lawrence Erlbaum, 1987.

[292] G. Syswerda, “Uniform Crossover in Genetic Algorithms,” in *Proceedings of the 3rd International Conference on Genetic Algorithms*, (J. D. Schaffer, ed.), pp. 2–9, Morgan Kaufmann, 1989.

[293] G. Syswerda, “A Study of Reproduction in Generational and Steady State Genetic Algorithms,” in *Foundations of Genetic Algorithms*, (G. J. E. Rawlings, ed.), pp. 94–101, San Mateo: Morgan Kaufmann, 1991.

[294] È. Taillard, “Tabu Search,” in *Local Search in Combinatorial Optimization*, (E. H. L. Aarts and J. K. Lenstra, eds.), ch. 1, pp. 1–17, Wiley, 1997.

[295] É. D. Taillard, “Robust Taboo Search for the Quadratic Assignment Problem,” *Parallel Computing*, vol. 17, pp. 443–455, 1991.

[296] É. D. Taillard, “Comparison of Iterative Searches for the Quadratic Assignment Problem,” *Location Science*, vol. 3, pp. 87–105, 1995.

[297] É. D. Taillard, “FANT: Fast Ant System,” Tech. Rep. IDSIA-46-98, IDSIA, Lugano, Switzerland, 1998.

[298] É. D. Taillard and L. M. Gambardella, “Adaptive Memories for the Quadratic Assignment Problem,” Tech. Rep. IDSIA-87-97, IDSIA, Lugano, Switzerland, 1997.

[299] G. Tao and Z. Michalewicz, "Inver-over Operator for the TSP," in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature - PPSN V*, (A.-E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, eds.), pp. 803–812, Springer, 1998.

[300] D. M. Tate and A. E. Smith, "A Genetic Approach to the Quadratic Assignment Problem," *Computers and Operations Research*, vol. 22, no. 1, pp. 73–83, 1995.

[301] N. L. J. Ulder, E. H. L. Aarts, H. J. Bandelt, P. J. M. van Laarhoven, *et al.*, "Genetic Local Search Algorithms for the Traveling Salesman Problems," in *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, (H. P. Schwefel and R. Männer, eds.), (Dortmund, Germany), pp. 109–116, Springer-Verlag, Berlin, Germany, 1-3 Oct. 1991.

[302] C. L. Valenzuela, "Evolutionary Divide and Conquer (II) for the TSP," in *Proceedings of the Genetic and Evolutionary Computation Conference*, (W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, eds.), pp. 1744–1749, Morgan Kaufmann, 1999.

[303] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications.* Kluwer Academic Publishers, 1987.

[304] T. E. Vollmann and E. S. Buffa, "The Facilities Layout Problem in Perspective," *Management Science*, vol. 12, no. 10, pp. 450–468, 1966.

[305] T. Walters, "Repair and Brood Selection in the Traveling Salesman Problem," in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature - PPSN V*, (A.-E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, eds.), pp. 813–822, Springer, 1998.

[306] E. D. Weinberger, "Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference," *Biological Cybernetics*, vol. 63, pp. 325–336, 1990.

[307] E. D. Weinberger, "Local Properties of Kauffman's N-k model: A tunably Rugged Energy Landscape," *Physical Review A*, vol. 44, no. 10, pp. 6399–6413, 1991.

[308] E. D. Weinberger, "NP Completeness of Kauffman's N-k Model, A Tuneable Rugged Fitness Landscape," Tech. Rep. 96-02-003, Santa Fe Institute, Santa Fe, New Mexico, 1996.

[309] E. D. Weinberger and P. F. Stadler, "Why Some Fitness Landscapes are Fractal," *J. Theor. Biol.*, vol. 163, pp. 255–275, 1993.

[310] D. Whitley, V. S. Gordon, and K. Mathias, "Lamarckian Evolution, The Baldwin Effect and Function Optimization," in *Parallel Problem Solving From Nature – PPSN III*, (Y. Davidor and H.-P. Schwefel, eds.), pp. 6–15, Springer, 1994.

[311] D. Whitley, T. Starkweather, and D. Fuquay, "Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 133–140, Morgan Kaufman, 1989.

[312] D. H. Wolpert and W. G. Macready, “No Free Lunch Theorems for Optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[313] S. Wright, “The Roles of Mutation, Inbreeding, Crossbreeding, and Selection in Evolution,” in *Proceedings of the Sixth Congress on Genetics*, p. 365, 1932.

[314] T. Yamada and R. Nakano, “Scheduling by Genetic Local Search with Multi-Step Crossover,” in *Proceedings of the 4th Conference on Parallel Problem Solving from Nature - PPSN IV*, (H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), pp. 960–969, Springer, 1996.

[315] H.-J. Zimmermann, *Methoden und Modelle des Operations Research. Für Ingenieure, Ökonomen und Informatiker. (Methods and models of operations research. For engineers, economists and computer scientists).* Friedr. Vieweg & Sohn, Braunschweig, 1987.