Combinatorial Optimization and the Analysis of Randomized Search Heuristics

Dissertation
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr. Ing.)
der Technischen Fakultät
der Christian-Albrechts-Universität
zu Kiel

Frank Neumann

Kiel
2006

1. Gutachter:	Prof. Dr. Rudolf Berghammer
2. Gutachter:	Prof. Dr. Ingo Wegener
3. Gutachter:	Priv.-Doz. Dr. Benjamin Doerr

Tag der mündlichen Prüfung: 19. Juli 2006

Abstract

Randomized search heuristics have widely been applied to complex engineering problems as well as to problems from combinatorial optimization. We investigate the runtime behavior of randomized search heuristics and present runtime bounds for these heuristics on some well-known combinatorial optimization problems. Such analyses can help to understand better the working principle of these algorithms on combinatorial optimization problems as well as help to design better algorithms for a newly given problem. Our analyses mainly consider evolutionary algorithms that have achieved good results on a wide class of NP-hard combinatorial optimization problems. We start by analyzing some easy single-objective optimization problems such as the minimum spanning tree problem or the problem of computing an Eulerian cycle of a given Eulerian graph and prove bounds on the runtime of simple evolutionary algorithms. For the minimum spanning tree problem we also investigate a multi-objective model and show that randomized search heuristics find minimum spanning trees easier in this model than in a single-objective one. Many polynomial solvable problems become NP-hard when a second objective has to be optimized at the same time. We show that evolutionary algorithms are able to compute good approximations for such problems by examining the NP-hard multi-objective minimum spanning tree problem. Another kind of randomized search heuristic is ant colony optimization. Up to now no runtime bounds have been achieved for this kind of heuristic. We investigate a simple ant colony optimization algorithm and present a first runtime analysis. At the end we turn to classical approximation algorithms. Motivated by our investigations of randomized search heurisitics for the minimum spanning tree problem, we present a multi-objective model for NP-hard spanning tree problems and show that the model can help to speed up approximation algorithms for this kind of problems.

Contents

Chapter 1

Introduction

The subject of this thesis is the runtime analysis of randomized search heuristics on different combinatorial optimization problems. Randomized search heuristics are general-purpose algorithms which often yield good solutions for problems whose structure is not known very well. Evolutionary algorithms (EAs) belonging to this class of algorithms have become quite popular since the '60s of the last century. Their main advantage is the fast application to different problem domains. Using an appropriate encoding for a given problem and some standard variation operators together with a fitness function for the considered problems, one can often come up with good solutions for the problem. Evolutionary algorithms have widely been applied to complex engineering problems as well as to problems from combinatorial optimization. In the case of a complex engineering problem the structure of the problem is often not known. Then the quality of a certain parameter setting can often only be evaluated by experiments or simulations. Such problems are considered in the field of black-box optimization, where the value of a parameter setting can only be given after having executed some experiments or simulations. EAs have shown to be very successful on many problems from black-box optimization. In the case of combinatorial optimization, often much more is known about the structure of a given problem and the function to be optimized can be given and analyzed. Nevertheless it is often difficult to obtain good solutions for such problems, especially if the problem is new and there are not enough resources (such as time, knowledge, money) to design specific algorithms for the given problem. In this case, the application of randomized search heuristics often yields satisfying results. Evolutionary algorithms and all other randomized search heuristics investigated in this thesis have shown to be very successful in obtaining good solutions, especially for new NP-hard optimization problems.

In contrast to the work done in the classical algorithm community, the work on evolutionary computation is mainly driven by experiments. A lot of work has been done in considering different encodings for some popular problems like NP-hard variants of the minimum spanning tree problem. The advantage of the newly created algorithm is then shown by reporting the results of experiments done on random graphs or some benchmark instances. Until the '90s of the last century the theoretical work in the area of evolution-

ary computation was concentrated on showing that an algorithm converges to an optimal solution after a finite number of steps. On the other hand it has been considered what happens in one iteration of the algorithms. Although these are interesting investigations, both aspects do not allow to give upper or lower bounds on the runtime of an evolutionary algorithm for a considered problem.

Runtime analyses of randomized search heuristics have to consider problems where the function to be optimized can at least be covered analytically. As explained before, this is often not possible for complex engineering problems. Therefore, we consider combinatorial optimization problems, which seem to be natural, but non-trivial examples where evolutionary algorithms have been applied. The rigorous analysis of evolutionary algorithms with respect to their runtime behavior is a relatively new research area. The first theoretical result on the runtime of an evolutionary algorithm has been given by Mühlenbein (1992). He has presented an upper bound on the expected runtime of the simplest evolutionary algorithm, called the (1+1) EA. The function considered is the simplest non-trivial pseudo-boolean function called OneMax which counts the number of ones in a given bitstring.

Since the mid '90s more rigorous results on the runtime of the (1+1) EA for different kinds of pseudo-boolean functions have been obtained. The first step was a much simpler proof for the Mühlenbein result and a generalization of the given bound to linear pseudo-boolean functions done by Droste, Jansen, and Wegener (2002). Considering different pseudo-boolean functions, the main aim was to show the behavior of EAs in different situations. Together with these results, many techniques have been developed that are very useful to analyze more complicated EAs as well as the behavior of randomized search heuristics on more natural problems.

In 2002, Scharnow, Tinnefeld and Wegener (2002) obtained the first results on the runtime of evolutionary algorithms on combinatorial optimization problems. Other, hopefully interesting, results are the subject of this thesis. We study the runtime behavior of simple randomized search heuristics on such well-known problems as the problem of computing Eulerian cycles, or easy and difficult spanning tree problems. To get an understanding how these heuristics work on combinatorial optimization problems, in a first step some of the best-known polynomially solvable problems have to be considered. We can not hope to beat the best problem-specific algorithms for such problems. But it is interesting to examine the runtime of these randomized search heuristics on these natural problems and such analyses can give hints on the behavior of EAs on NP-hard problems. Other important results have been achieved by Giel and Wegener (2003, 2004) and Witt (2005). Giel and Wegener have considered the (1+1) EA for the computation of maximum matchings. Witt (2005) has given a worst-case and an average-case analysis for the NP-hard partition problem.

The mentioned combinatorial optimization problems are single-objective ones. EAs

have especially shown to be successful to obtain good results for multi-objective problems. In contrast to single-objective optimization, here one searches for a set of solutions instead of a single one. Multi-objective combinatorial optimization problems have not been such extensively studied as single-objective ones in the algorithm community. Therefore, less is known about the structure of such problems and general search heuristics often obtain satisfying results. In particular, EAs seem to be, in a natural way, a good choice as they evolve the population, which is a set of possible solutions, during the optimization process. The first result on the runtime of a multi-objective evolutionary algorithm (MOEA) has been given by Laumanns, Thiele, Zitzler, Welzl, and Deb (2002). They considered a MOEA that searches locally. Later, Giel (2003) investigated a MOEA which has the opportunity to sample each search point of the search space with a positive probability. Both analyses are on artificial functions. We will analyze simple MOEAs for combinatorial optimization problems.

In Chapter 2, we introduce some preliminaries for considering combinatorial optimization problems. Here, we first take a view on single-objective optimization and discuss afterwards the basic issues for considering multi-objective optimization problems. In Chapter 3 we give an introduction to different kinds of randomized search heuristics. The investigations in this thesis will mainly consider evolutionary algorithms, so we start by presenting the different approaches developed in this field. Another kind of randomized search heuristic that has become quite popular especially for combinatorial optimization problems is ant colony optimization (ACO). We discuss the important issues of ACO. To give a more complete picture we present other popular variants of randomized search heuristics afterwards. In Chapter 4, we introduce the randomized search heuristics that are subject to the analyses throughout this thesis and discuss some popular methods for the runtime analysis in Chapter 5.

In Chapter 6, we consider the Eulerian cycle problem, presented by Euler in 1736, which can be seen as the first problem in graph theory. In the Eulerian cycle problem an undirected graph is given and one searches for a permutation of the edges such that a tour is created where each edge is used exactly once. Graphs for which such a tour exist are called Eulerian. We investigate simple EAs that work with different mutation operators and show that they are able to compute an Eulerian cycle of a given Eulerian graph in expected polynomial time if a jump operator is used for mutation. Altering the mutation operator from jumps to exchanges, we present an example, where the expected runtime increases drastically i. e. from polynomial to exponential.

In Chapter 7, we investigate the minimum spanning tree problem. A main part of this thesis will deal with considerations for different kinds of spanning tree problems. The well-known problem of computing minimum spanning trees can be solved in polynomial time using the greedy approaches due to Kruskal and Prim (see e. g. Cormon, Leiserson, Rivest, and Stein (2001)). We do not hope to beat such special-purpose algorithms by randomized search heuristics. Nevertheless, evolutionary algorithms have obtained good results

on NP-hard variants of the minimum spanning tree problem. Therefore, we think that it is important to get an understanding how these heuristics work on this basic problem and analyze simple randomized search heuristics with respect to their runtime behavior. We give upper and lower bounds on the runtime of simple EAs until they have computed a minimum spanning tree.

In Chapter 8, we switch to ant colony optimization. In contrast to many successful applications, the theoretical foundation of this randomized search heuristic is rather weak. Building up such a theory is demanded to understand how these heuristics work as well as to come up with better algorithms for certain problems. Up to now, only convergence results have been obtained showing that optimal solutions can be reached in a finite amount of time. We present the first runtime analysis of a simple ACO algorithm that transfers many rigorous results with respect to the runtime of a simple evolutionary algorithm, including the ones presented in Chapter 7, to the considered algorithm. In addition, we examine the choice of the evaporation factor, which is a crucial parameter in such an algorithm, in greater detail and analyze its effect with respect to the runtime.

In Chapter 9, we return to evolutionary algorithms and consider a simple MOEA for a multi-objective combinatorial optimization problem. Many classical polynomially solvable problems become NP-hard when a second objective has to be optimized at the same time. In this case one is interested in good approximations. In the case of minimum spanning trees, the problem gets NP-hard if at least two weight functions have to be optimized simultaneously. This is the problem of computing multi-objective minimum spanning trees. Here one searches for spanning trees such that the weight with respect to one objective function can only be decreased by increasing the weight of another objective function. Such solutions are called Pareto optimal and the set of objective vectors belonging to the set of all Pareto optimal solutions is called the Pareto front. We give upper bounds until a simple MOEA has achieved a good approximation of the Pareto front for the multi-objective minimum spanning tree problem.

In Chapter 10, we consider another important question. Is it possible that a randomized search heuristic works better in a multi-objective model than in a single-objective one for a given single-objective optimization problem? Usually multi-objective optimization is considered as more (at least as) difficult as single-objective optimization. But one should not forget that the fitness vector of the different objectives to optimize can give better hints for the optimization process of evolutionary algorithms. We consider a multi-objective model of the minimum spanning tree problem and show that this model leads to a better optimization process than in the case of the single-objective model examined in Chapter 7.

In Chapter 11, we switch to classical approximation algorithms and examine whether the approach of using a multi-objective model for a single-objective optimization problem can also lead to faster approximation algorithms. We consider spanning tree problems with additional degree constraints. These problems are NP-hard as they include the problem of

finding a Hamiltonian path in a given graph. We extend the multi-objective model used for evolutionary algorithms and the minimum spanning tree problem in Chapter 10 and show that this view on the considered problems gives new insights to come up with faster approximation algorithms than the up to now best ones.

Many people have supported me during my work on this thesis. First, I want to thank Rudolf Berghammer for supporting my research on randomized search heuristics within his group. I am grateful to Ingo Wegener for many discussions on the analysis of evolutionary algorithms and for his cooperation during the last years. I am in debt to Marco Laumanns for many conversations on evolutionary algorithms as well as on multi-objective optimization, and to Carsten Witt for several discussions on ACO algorithms as well as for proof-reading. For the good working atmosphere and proof-reading I want to thank Britta Kehden. Last but not least, I thank my wife Aneta and our daughter Michelle for their support and love.

Part I

Basics

Chapter 2

Combinatorial Optimization

In this chapter we introduce the basic questions that are considered in the field of combinatorial optimization. Often problems are investigated where the function value of one single goal function has to be optimized. We refer to such problems as single-objective optimization problems. In the case of single-objective optimization one searches for one single solution that should be an optimal one or in the case of difficult problems has a certain approximation quality. As most of the results on combinatorial optimization problems refer to single-objective problems, we discuss the basic concepts and approaches to solve these problems in Section 2.1. In the case that at least two goal functions have to be optimized at the same time we are dealing with multi-objective optimization problems. In this case one has to apply other techniques as one usually searches for a set of solutions instead of a single one. We discuss this situation in Section 2.2.

2.1 Single-objective optimization

Typical textbooks on algorithms (see e. g. Cormen, Leiserson, Rivest, and Stein (2001)) focus on single-objective optimization problems. Here one single goal function is considered whose function value should be optimized. Optimization problems can be divided naturally in two categories. The first category consists of problems with continuous variables. In the case of discrete variables we call an optimization problem combinatorial. Speaking of combinatorial optimization problems most people have "natural" examples in mind such as the computation of minimum spanning trees or a shortest path between two vertices in a given graph.

In a combinatorial optimization problem one either aims at minimizing or maximizing a given objective function. A problem is a general question to be answered such as finding a shortest path in a given graph. Such a problem has usually a set of input parameters. In the case of the shortest path problem there is a set of vertices and a set of weighted edges that tell whether two vertices are connected and give the cost of such a connection. An instance of a problem is given by the problem together with a specified parameter setting.

More formally a combinatorial optimization problem can be defined as a triple (S, f, Ω), where S is a given search space, f is the objective function which should be either maximized or minimized and Ω is the set of constraints that have to be fulfilled to obtain feasible solutions. The goal is to find a globally optimal solution which is in the case of a maximization problem a solution s^* with the highest objective value that fulfills all constraints. Similarly in the case of minimization problems one tries to achieve a smallest objective value under the condition that all constraints are fulfilled. In contrast to the description of the problem which is usually short, the search space is most of the time exponential in the problem dimension. In addition for a lot of combinatorial optimization problems one can not hope to come up with an algorithm that produces for all problem instances an optimal solution within a time bound that is polynomial in the problem dimension.

Throughout this work we consider combinatorial optimization problems on graphs. A directed graph G is a pair $G = (V, E)$, where V is a finite set and E is a binary relation on V. The elements of V are called vertices. E is called the edge set of G and its elements are called edges. We use the notation $e = (u, v)$ for an edge in a directed graph. Note that self-loops that are edges of the kind (u, u) are possible. In an undirected graph $G = (V, E)$ no self-loops are possible. The edge set E consists of unordered pairs of vertices in this case, and an edge is a set $\{u, v\}$ consisting of two distinct vertices $u, v \in V$. Note that one can think of an undirected edge $\{u, v\}$ as two directed edges (u, v) and (v, u).

If (u, v) is an edge in a directed graph $G = (V, E)$ we say that v is adjacent to vertex u. This leads to the representation of graphs by adjacency matrices and will be discussed later in greater detail. A path of length k from a vertex v_0 to a vertex v_k in a graph $G = (V, E)$ is a sequence $v_0, v_1, \ldots, v_k$ of vertices such that $(v_{i-1}, v_i) \in E$, $1 \leq i \leq n$, holds. Note that a path implies a sequence of directed edges. Therefore, it is sometimes useful to denote a path $(v_0, v_1, \ldots, v_k)$ by its sequence of directed edges $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$.

The performance measure most widely used to analyze algorithms is the time an algorithm spends until it presents its final answer. Time is expressed in terms of number of elementary operations such as comparisons or branching instructions (see e. g. Papadimitriou and Steiglitz (1998)). The time an algorithm needs to give the final answer is analyzed with respect to the input size. The input of a combinatorial optimization problem is a graph, a set of integers, and so on. To submit it to a computer it has to be encoded or represented as a sequence of symbols of a finite alphabet. The size of the input is the length of this sequence, that is, the number of symbols in it. In this work we are dealing with combinatorial optimization problems where the task is to optimize different properties for a given undirected graph. Consider an undirected graph $G = (V, E)$ with n vertices and m edges. This graph can be represented by an adjacency matrix $A_G = [a_{ij}]$ where $a_{ij} = 1$ if $(v_i, v_j) \in E$ and $a_{ij} = 0$ otherwise. Note that the adjacency matrix of a given undirected graph is symmetric. An undirected graph may have up to $\binom{n}{2} = \Theta(n^2)$ edges. However if we are considering sparse graphs the number of edges is far less than $\binom{n}{2}$. Therefore the representation as adjacency lists seems to be more suitable in this case. For each vertex $v \in V$ we record a set $A(v) \subseteq V$ of vertices that are adjacent to it. The size

of the representation is given by the sum of the length of lists. As each edge contributes 2 to this total length we have to write down $2m$ elements. Another factor which effects the total length of the representation is how to encode the vertices. Our alphabet has finite size. Assume the alphabet is the set $\{0, 1\}$. Therefore we need $\Theta(\log n)$ bits to encode one single vertex. This implies that we need $\Theta(m \log n)$ bits (or symbols) to represent the graph G. In practice we say that a graph G can be encoded in $\Theta(m)$ space which seems to be a contradiction to the previous explanation. The reason is that computers treat all integers in their range the same. Here the same space is needed to store small integers as for example 5 or big integers as for example 3^{12}. We assume that graphs are considered where the number of vertices is within the integer range of the computer. This range is is in most cases 0 to 2^{31} which means that integers are represented by 32 bits. Therefore $\Theta(m)$ is a reasonable approximation of the size of a graph and analyzing graph algorithms with respect to m is accepted in practice. In most cases both parameters n and m are taken into account when analyzing the complexity of a graph algorithm.

Another issue that has to be considered when representing the input of a given problem is that some combinatorial optimization problems such as the traveling salesperson problem or the minimum spanning tree problem have parameters that consist of integers. Then these problems involve operations such as addition and comparison of integers. It may be the case that the integers are so large that such operations can not be handled by the finite work length of our hypothetical computer. Then special techniques have to be used to carry out such operations. Such techniques usually require an amount of time that grows approximately as the logarithm of the integers. For simplicity each of such an operation is often considered as an elementary step.

An important issue that comes up when considering combinatorial optimization problems is the classification of difficult problems (see e. g. Papadimitriou (1994) or Wegener (2005a)). To distinguish between easy and difficult problems one considers the class of problems that are solvable by a deterministic Turing machine in polynomial time and problems that are solvable by a nondeterministic Turing machine in polynomial time. The complexity class belonging to the first kind of problems is called P and the other one is called NP. It is easy to see that $P \subseteq NP$ holds, and widely assumed that $P \neq NP$. For a lot of important combinatorial optimization problems it has been shown that they are at least as difficult as the most difficult problems in NP. Such problems are called NP-hard. As we assume that $P \neq NP$ holds, we can not hope to come up with an algorithm that finds for each instance of such a problem an optimal solution in polynomial time.

Considering graph algorithms where we can bound the runtime by a polynomial in n and m we obviously get a polynomial time algorithm. We have to be careful when the input includes numbers. Let $N(I)$ be the largest integer that appears in the input. An algorithm A is called pseudo-polynomial if it is polynomial in the input size $|I|$ and $N(I)$. Note that $N(I)$ can be encoded by $\Theta(\log(N(I)))$ bits. Therefore a function that is polynomial in $|I|$ and $N(I)$ is not necessarily polynomial in the input size. Often the input consists of small

integers. In the case that $N(I)$ is bounded by a polynomial in $|I|$, A is a polynomial time algorithm.

The classical approach to deal with NP-hard problems is to search for good approximation algorithms (see e. g. Hochbaum (1997) or Vazirani (2003)). These are algorithms that run in polynomial time but guarantee that the produced solution is within a given ratio of an optimal one. Such approximation algorithms can be of a totally different kind for different optimization problems. In the case of the NP-hard bin packing problem even simple greedy heuristics work very well whereas in the case of more complicated scheduling problems often methods based on linear programming are used. In Chapter 8 we will consider approximation algorithms that are based on local search procedures. The analysis of one of these algorithms is based on a linear programming formulation of the considered problem.

Another approach to solve NP-hard problems is to use sophisticated exact methods that have in the worst case an exponential runtime. The hope is that such algorithms produce for interesting problem instances good results in a small amount of time. A class of algorithms that tries to come up with exact solutions is branch and bound. Here the search space is shrunk during the optimization process by computing lower bounds on the value of an optimal solution in the case that we are considering maximization problems. The hope is to come up in a short period of time with a solution that matches such a lower bound. In this case an optimal solution has been obtained.

A crucial point when considering combinatorial optimization problems and randomized search heuristics that search more or less locally is the consideration of the neighborhood of the current search point. Let $s \in S$ be a search point in a given search space. The neighborhood is defined by a mapping $N\colon S \to 2^S$. In the case that we are considering combinatorial optimization problems from the search space $\{0,1\}^n$ the neighborhood can be naturally defined by all solutions having at most Hamming distance k to the current solution s. The parameter k determines the size of the neighborhood from which the next solution is sampled. Choosing a small value k, e. g. $k = 1$, such a heuristic may get stuck in local optima. If the value of k is large (in the extreme case $k = n$) and all search points of the neighborhood are chosen with the same probability, the next solution will be somehow independent of s. This leads to randomized search heuristics that nearly behave as choosing in each step a search point uniformly at random from $\{0,1\}^n$. In this case the randomized search heuristic do not take the previously sampled function values into account and the search can not be directed into "good" regions of the considered search space.

2.2 Multi-objective optimization

Many problems in computer science ask for solutions with certain attributes or properties that can be expressed as functions mapping possible solutions to scalar numeric values. The usual optimization approach is to take these attributes as constraints to determine the feasibility of a solution, while one of them is chosen as an objective function to determine the preference order of the feasible solutions. In the minimum spanning tree problem, as a simple example, constraints are imposed on the number of connected components (one) and the number of cycles (zero) of the chosen subgraph, while the total weight of its edges is the objective to be minimized. A more general approach is multi-objective optimization (see, e.g. Ehrgott (2005) or Papadimitriou and Yannakakis (2000)), where several attributes are employed as objective functions and used to define a partial preference order of the solutions, with respect to which the set of minimal (maximal) elements is sought. Most of the best known single-objective polynomial solvable problems like shortest path or minimum spanning tree become NP-hard when at least two weight functions have to be optimized at the same time. In this sense, multi-objective optimization is considered as more (at least as) difficult than (as) single-objective optimization.

In the case of multi-objective optimization the objective function $f = (f_1, \ldots, f_k)$ is vector-valued, i. e., $f\colon S \to \mathbb{R}^k$. Since there is no canonical complete order on $\mathbb{R}^k$, one compares the quality of search points with respect to the canonical partial order on $\mathbb{R}^k$, namely $f(s) \leq f(s')$ iff $f_i(s) \leq f_i(s')$ for all $i \in \{1, \ldots, k\}$. A Pareto optimal search point s is a search point such that (in the case of minimization problems) $f(s)$ is minimal with respect to this partial order and all $f(s'), s \in S$. Again there can be many Pareto optimal search points but they do not necessarily have the same objective vector. The Pareto front, denoted by F, consists of all objective vectors $y = (y_1, \ldots, y_k)$ such that there exists a search point s where $f(s) = y$ and $f(s') \leq f(s)$ implies $f(s') = f(s)$. The Pareto set consists of all solutions whose objective vector belongs to the Pareto front. The problem is to compute the Pareto front and for each element y of the Pareto front one search point s such that $f(s) = y$. We sometimes say that a search point s belongs to the Pareto front which means that its objective vector belongs to the Pareto front.

As in any case of optimization problems one may be satisfied with approximate solutions. This can be formalized as follows. For each element y of the Pareto front we have to compute a solution s such that $f(s)$ is close enough to y. Close enough is measured by an appropriate metric and an approximation parameter. In the single-objective case one switches to the approximation variant if exact optimization is too difficult. The same reason may hold in the multi-objective case. There may be another reason. The size of the Pareto front may be too large for exact optimization.

The Pareto front F may contain exponentially many objective vectors. Papadimitriou and Yannakakis (2000) have examined how to approximate the Pareto front for different multi-objective combinatorial optimization problems. W. l. o. g. they have considered the

task to maximize all objective functions. Given an instance I and a Parameter $\epsilon > 0$ they have examined how to obtain an ϵ-approximate Pareto set. This is a set of solutions X with the property that there is no solution s' such that for all $s \in X$ $f_i(s') \geq (1+\epsilon) \cdot f_i(s)$ holds for at least one i. Their results show that there exists for each multi-objective optimization problem an ϵ-approximate Pareto set X of solutions that is polynomially bounded in $|I|$ and $1/\epsilon$.

Papadimitriou and Yannakakis show that there exists an algorithm which constructs such a set X which is polynomially bounded in $|I|$ and $1/\epsilon$ if and only if the corresponding gap problem problem can be solved. Given an instance I of the considered problem and a vector $(b_1, \ldots, b_k)$ the gap problem consists of either presenting a solution s with $f_i(s) \geq b_i$, $1 \leq i \leq k$, or answering that there is no solution s' with $f_i(s') \geq (1+\epsilon) \cdot b_i$, $1 \leq i \leq k$. In the case of some multi-objective optimization problems (e. g. the multi-objective variants of the minimum spanning tree problem and the shortest path problem) such a set can also be computed within a time bound that is polynomial in $|I|$ and $1/\epsilon$. Algorithms with such properties constitute a multi-objective fully polynomial time approximation scheme (FPTAS) which is the best we can hope for when dealing with NP-hard problems.

Especially in the case of multi-objective optimization, EAs seem to be a good heuristic approach to obtain a good set of solutions. EAs have the advantage that they work at each time step with a set of solutions called the population. This population is evolved to obtain a good approximation of the Pareto front. In most cases the quality of a newly designed EA is evaluated by experiments. In Chapter 9 we will consider the multi-objective minimum spanning tree problem and examine which parts of the Pareto front can be obtained by a simple EA in expected pseudo-polynomial time.

The question arises whether working in the more general framework of multi-objective optimization can lead to better understanding of a given problem or help to design more efficient algorithms for single-objective problems. Note that many single objective problems have additional constraints that classify feasible and unfeasible solutions of the given search space. Such constraints can be relaxed such that additional objectives have to be optimized. Then the set of minimal elements contains the solution of the corresponding constrained single-objective problem. This has already been considered in the average case analysis of a well-known algorithm for the 0/1 knapsack problem. Beier and Vöcking (2003) have considered different input distributions for this problem and shown that the number of minimal elements in the objective space is polynomially bounded. This implies that the well-known algorithm of Nemhauser and Ullmann (1969) has an expected polynomial runtime for these distributions. In Chapter 7 and 10 we will analyze the runtime of randomized search heuristics for the computation of minimum spanning trees. Our results show that randomized search heuristics find minimum spanning trees more easier in a multi-objective model than in a single-objective one. A welcome byproduct of a successful multi-objective approach is to obtain more information (a set of minimal elements instead of only one specific element in it) with even less computational effort. This approach can also be used

to obtain faster approximation algorithms for NP-hard problems. In Chapter 11 we will show that a multi-objective formulation of NP-hard spanning tree problems can help to come up with faster approximation algorithms for these problems.

Chapter 3

Randomized Search Heuristics

In this chapter, we give an introduction into the field of randomized search heuristics. Mainly we will consider randomized heuristics belonging to the field of evolutionary computation throughout this work. These algorithms are inspired by the evolution process in nature and follow Darwin's principle of the survival of the fittest. We take a closer look at the different approaches developed in this field in Section 3.1. Another kind of bio-inspired randomized search heuristic is ant colony optimization (ACO), which will be introduced in Section 3.2. Here solutions for a given problem are constructed by walks of ants on a so-called construction graph. To give a more complete picture we describe other popular variants in Section 3.3.

A randomized search heuristic is a problem-independent algorithm to solve problems from a considered search space although it might have modules that are adjusted to the considered problem or are combined with problem-dependent algorithms. The independence from the considered problem distinguishes randomized search heuristics from problem-dependent algorithms developed and analyzed in the classical algorithm community. In contrast to the classical approach to algorithms where one designs an algorithm with the task to prove bounds on the runtime and/or approximation quality in mind, randomized search heuristics are general purpose algorithms. Assuming that one considers different problems from the same search space, e.g. $\{0,1\}^n$, a randomized search heuristic is usually applicable to each of these problems. Their easy adaptation to different problems usually has to be paid by the disadvantage that the algorithm is often not rigorously analyzed with respect to its runtime and/or approximation quality. Due to the No Free Lunch Theorem (Wolpert and Macready (1997)) each algorithm behaves on average the same on all possible functions $f : S \to R$ where S is an arbitrary finite search space and R is the finite set of possible function values. This also implies that no randomized search heuristic behaves on the average better than blind random search, where in each step a solution is drawn uniformly at random from the considered search space. This should make clear that an analysis of these algorithms with respect to their runtime makes only sense for specific classes of functions or specific classes of problems.

In the general approach the only problem-dependent component of the algorithm is the fitness function that guides the search. This function is the only part of such an algorithm that has to be adjusted to the considered problem. Therefore, we get algorithms that can be implemented very easily and adjusted quickly to similar problems. As already mentioned randomized search heuristics are not designed to prove special properties on the runtime or approximation quality. This makes a rigorous analysis of such algorithms somehow more difficult than the analysis of an algorithm that has been designed in a special way to prove properties such as the runtime or approximation quality of the algorithm.

We start with a general description of randomized search heuristics. Given a search space S the aim is to optimize a considered function $f : S \to R$ where R is the set of all possible function values. A randomized search heuristic working in a given search space S under the consideration of a function f chooses the first search point s_1 with respect to a probability distribution on S that may be determined by a heuristic. After that the function value $f(s_1)$ is computed. The search point s_t is chosen due to a probability distribution that can depend on the previous sampled search points $s_1, \ldots, s_{t-1}$ and their function values. The process is iterated until a stopping criterion has been fulfilled.

This description covers all important approaches such as evolutionary algorithms, ant colony optimization, randomized local search, the Metropolis algorithm, and simulated annealing.

3.1 Evolutionary algorithms

Evolutionary algorithms (EAs) have become quite popular since the mid '60s of the last century. Inspired by the evolution process in nature they try to solve problems by evolving sets of search points such that satisfying results are obtained. A lot of tasks that have been solved by EAs lie in the field of real-world applications. In real-world applications the function to be optimized is often unknown and function values can only be obtained by experiments. Often these experiments cause high costs or need a large amount of time. Therefore, the main aim is to minimize the number of function evaluations until a satisfying result has been obtained.

The main difference between evolutionary algorithms and local search procedures or simulated annealing is that evolutionary algorithms usually work at each time step with a set of solutions which is called the population of an EA. This population produces a set of solutions called the offspring population by some variation operators such as crossover or mutation. After that a new population is created by selecting individuals from the parent and offspring population due to the fitness function f. We consider discrete search spaces throughout this work. In this case another important issue is that evolutionary algorithms often have a positive probability of sampling each search point of the given search space in the next step. In the case of local search and simulated annealing this is usually not

the case. Here the search points that can be constructed in the next step depend on the current solution and the neighborhood defined for the search process.

We want to take a look at the different modules of an EA. The first important issue is representation. Solutions can be represented in different ways. A good example are the different representations of spanning trees. For a given undirected connected graph with n vertices and m edges the most natural representation seems to be a set of $n-1$ edges such that the graph is connected. This is known as the representation of spanning trees by edge sets (see Raidl and Julstrom (2003)). More general is to represent them as bitstrings of length m, where each bit corresponds to an edge which is included in the solution if the bit is set to 1 and excluded otherwise. In this case information to obtain a connected graph or a spanning tree has to be included into the fitness function. We will use this representation later for the analysis of EAs on the minimum spanning tree problem. There it turns out that guiding such an algorithm to compute connected graphs or spanning trees is a minor term in the overall complexity.

Spanning trees can also be represented by Prüfer numbers. A Prüfer number consists of $n-2$ node identifiers which determine a spanning tree. This number can be decoded by an algorithm into a corresponding spanning tree and a spanning tree can be encoded into a Prüfer number using a complementary algorithm. The disadvantage is that small changes in the Prüfer number can result in a totally different spanning tree. Therefore Prüfer numbers are a poor representation of spanning trees when using an EA (see Gottlieb, Julstrom, Raidl, and Rothlauf (2001)). This is not the case when edge sets are considered. If the set of edges is changed by one edge then the two spanning trees have of course $n-2$ edges in common. It should be clear that this point of locality is important for the success of an EA. If only small changes lead to a completely different solution with also a fitness value that does not depend on the last sampled search point, the search cannot be directed into "good" regions of the search space.

An other issue of representation has recently been examined by Kehden and Neumann (2006). They have investigated whether representing the population of an EA by relations can speed up the computation time of an EA. It turns out that the computation time for one generation can be reduced for some problems from $\Theta(n^3)$ to $O(n^{2.376})$ using relational algebra if the population size is $\Theta(n)$ which is often the case. For larger population sizes it is also possible to obtain this speed-up and save a factor of $n^{0.624}$. The result is obtained by a relation-algebraic view on the evaluation process for some of the most important graph problems such as minimum vertex covers, maximum cliques, and maximum independent sets and makes use of the matrix multiplication algorithm by Coppersmith and Winograd (1990). In a recent work of Cohn, Kleinberg, Szegedy, and Umans (2005) it is conjectured that matrix multiplication can be done in time $O(n^2)$ which would imply this upper bound for the computation time for one generation when working with relations and populations of size $\Theta(n)$ for the mentioned problems.

Variation operators are important to construct new solutions. They have to be adjusted to the chosen representation. The most popular variation operators are mutation and crossover. In the case of mutation one single individual is altered, in a crossover operation at least two individuals produce new solutions. Often in a first step crossover is used to produce offsprings and these offsprings are additionally altered by a mutation operator. Throughout this work we will analyze EAs that use only mutation to obtain new solutions.

Nevertheless, to give a more complete picture, we also present some popular crossover operators for the search space $\{0,1\}^n$ and the representation of permutations. Crossover operators produce new search points by combining search points of the current population. We first take a look at the case where solutions are represented as bitstrings of length n. The most important crossover operators for individuals that are bitstrings of length n are uniform and k-point crossover, where usually $k \in \{1,2\}$ is chosen. Consider two individuals $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ that should produce a new solution $z = (z_1, \ldots, z_n)$ by a crossover operator. In the case of uniform crossover $Prob(z_i = x_i) = Prob(z_i = y_i) = 1/2$ if $x_i \neq y_i$ holds. Otherwise $z_i = x_i = y_i$ holds for the created child z. In the case of k-point crossover k positions in the two bitstrings are selected at random. Due to these positions the individuals are partitioned into different intervals, where the intervals are numbered due to their appearance in the bitstrings. The new individual z is formed by taking all entries of intervals with odd numbers from x and all entries of intervals with even numbers from y.

In the case of the representation of permutations it is a little bit more difficult to obtain sensible crossover operators. We assume that we are working with permutations consisting of n elements. Most crossover operators are applied to two parents P_1 and P_2 and produce two offspring O_1 and O_2. To give an impression how crossover operators for permutation problems are designed we consider the order crossover operator (OX-operator), which gets two parameters i and j, $1 \leq i, j \leq n$. W.l.o.g. we assume $i < j$. In a first step the elements of P_1 at positions $i+1, \ldots, j-1$ are copied into O_1 to the same positions. After that the remaining elements of P_2 are placed into O_1. This is done by examining P_2 from position j on in a circular way and placing the elements that up to now do not occur in O_1 at the next position where the positions $j, \ldots, n, 1, \ldots, i$ are considered one after another. In the same way the offspring O_2 is constructed starting by copying the elements between the positions i and j of P_2 into O_2.

We describe important mutation operators for the search space of binary strings and permutations of elements in the following. In the case of bitstrings of length n each bit is often flipped with a certain probability p, where $p = o(1)$ usually holds. It is necessary to choose p not too large to prevent the algorithm from sampling the next solution nearly uniformly at random from a very large neighborhood of the parent solution. In a lot of algorithms $p = 1/n$ is used such that on average 1 bit is flipped. In the case of permutations with n elements often jumps or exchange operations are used. Both operations get two parameters i and j, $1 \leq i, j \leq n$. Then a $jump(i,j)$ places the element at position i at position j and shifts the elements between i and j, including j, into the appropriate

direction. If $i < j$ the elements are shifted to the left and shifted to the right if $i > j$. An $exchange(i, j)$ places the element at position i to position j and the element at position j to position i. W. l. o. g. assume that $i < j$ holds for $exchange(i, j)$. Then this operation can be simulated by executing sequentially the two jump operations $jump(i, j)$ and $jump(j - 1, i)$. In contrast to this $\lfloor k/2 \rfloor$ exchange operations are needed to simulate $jump(i, j)$ if $|i - j| = k$ holds. Therefore the jump operator seems to be the more flexible one. We will see later that this can make the difference between a polynomial and an exponential expected runtime.

Selection methods are used to decide which individuals of the current population are used to produce offsprings. In addition they are used to decide which individuals from the parent and offspring population constitute the population of the next generation. A widely used selection method is fitness-proportional selection. We assume that the function f should be maximized and that all function values are positive. If the population contains μ individuals $x^1, \ldots, x^\mu$, then x^i has probability $f(x^i)/(\sum_{i=1}^{\mu} f(x_i))$ of being chosen in each selection step. Note, that this selection method allows to choose individuals more than once for a certain purpose. Therefore the population of the next generation may include duplicates even if the parent and offspring population before have contained only individuals that were pairwise distinct from each other. Another important method is tournament selection. Here tournaments of size $q \in \{1, \ldots, \mu\}$ are chosen. In each tournament q individuals compete against each other. The individuals that take place in a certain tournament are chosen uniformly at random from the population. In each tournament the individual with the highest fitness value wins the competition and is chosen for reproduction respectively the next generation. Two other important selection methods are $(\mu + \lambda)$- and (μ, λ)-selection. These two methods have their main application in evolution strategies. We will discuss the different approaches in evolutionary computation together with these two methods in the following.

The class of evolutionary algorithms covers historically different approaches to solve problems inspired by the evolution process in nature. The approaches differ by the search spaces that are considered and the variation operators used to produce new search points.

Evolution strategies (ES) (see Rechenberg (1973) or Schwefel (1981, 1995)) are used to solve continuous optimization problems. There usually a real-valued search space is considered. Mutation is the variation operator that is mainly used in ES. The most important strategies are called (μ, λ)- and $(\mu + \lambda)$-ES and differ from each other by the chosen selection method. In the case of a (μ, λ)-ES, the parent population has size μ and λ children are produced in one generation. The next parent population is created by choosing μ individuals from the offspring population. Note that in this case $\lambda >> \mu$ should hold as the parent population is not involved in the selection process. In contrast to this a $(\mu + \lambda)$ strategy considers both populations for the next parent population. After having created λ children, individuals from the parent and the offspring population are chosen due to their fitness values to build the parent population of the next generation.

Genetic algorithms (GAs) introduced by Holland (1975) work in discrete search spaces. Here bitstrings of length n are used to represent possible solutions. The other main difference to evolution strategies is that crossover is seen as the variation operator that has the main effect to get good solutions. Working with a population of size μ in each iteration μ children are produced by using crossover. Mutation is seen as the minor variation operator. If it takes place it is often applied to each child that has been produced by crossover. Then each bit is flipped with a certain probability p where often $p = 1/n$ is chosen. The major selection method for GAs is fitness proportional selection. This method is used to select the individuals that are used to obtain new solutions as well as to select the individuals from the parents and children to form the population of the next generation. Another variant is to produce only a few children in each iteration. In the extreme case one child is produced. This is known as the steady state GA. A lot of theoretical work for GAs has been concentrated on schemata. A schema fixes some positions in the bitstrings such that a search space of a smaller dimension is obtained. It is assumed that genetic algorithms combine schemata to obtain better ones. This implicitly assumes that the function which should be optimized is separable and leads to the building block hypothesis. This hypothesis says that functions are optimized by separating the variables and optimizing functions that depend on these partitions. It is assumed that such a partitioning is found by a GA and that the different blocks can be optimized in parallel. The problem is that even simple functions are not separable. Despite the fact that the schema theorem considers the behavior of a GA only in one step, the major lack is that the building block hypothesis has no clear formulation that can be verified or falsified.

Evolutionary programming (EP) (see e. g. Fogel, Ownes, and Walsh (1966)) considers a representation that is fit to the problem. This means that the different parameters that have to be optimized can have different codomains. The main variation operator is mutation which can be handled very flexibly, and EP makes usually no use of crossover operators. In a standard approach a parent population of size μ produces μ children by mutation. The new parent population consists of μ individuals from the parents and children that have been selected by a probabilistic selection method (e.g. fitness proportional selection). In the selection step it is important to ensure that a best individual of the parents and the children is integrated into the new parent population such that the best solution found will not get lost during the optimization process.

Genetic programming (GP) developed by Koza (1990) is an evolutionary computation approach that has become very popular in recent years. Instead of searching the considered search space one tries to construct good computer programs that solve the given task. Therefore, individuals are possible computer programs, usually represented as trees that represent expressions. These trees are evolved during the evolution process. Similar to the other approaches a set of computer programs constitutes a population, and a parent population creates an offspring population using crossover and mutation. The fitness of a given program is given by its performance with respect to the evaluation of some test cases.

To select individuals from the parents and the children for the new parent population often fitness-proportional selection is used.

3.2 Ant colony optimization

Ant colony optimization (ACO) is another bio-inspired approach to solve optimization problems. Introduced by Dorigo, Maniezzo, and Colorni (1991) it has especially shown to be successful to solve combinatorial optimization problems. A good overview over the different techniques used in this field is given in the book of Dorigo and Stützle (2004). In contrast to EAs where solutions are constructed from the current set of solutions, solutions are in this case obtained by random walks on a so-called construction graph which is usually a directed graph. ACO algorithms are inspired by the search of an ant colony for a common source of food. It has been noticed that ants find very quickly a shortest path to a source of food. The information about which way to take to get to the food is distributed between the ants by leaving an information, called pheromone, on the way an ant has taken. As longer paths to the source take much more time than shorter paths, shorter paths are more often visited. This implies larger pheromone values on shorter paths after a small amount of time.

These ideas are used to solve optimization problems. Solutions of a given problem are obtained by random walks of ants on a construction graph that has positive values, the pheromone values, on the edges. These values influence the random walks in the way that edges with large values have a larger probability of being traversed. In addition the model of ACO algorithms allows to include heuristic information to guide the random walks. This information additionally influences the probability which vertex to visit next in the random walk.

In an ACO algorithm each ant of the colony exploits the construction graph to search for an optimal solution. We assume that the ant colony is a set $A = \{a_1, \ldots, a_k\}$ of k ants. Each a_i has a memory that can be used to store information about the path it has followed so far. This memory can be used to build feasible solutions, compute a heuristic value η, evaluate the solution that has been found, and retrace the path backwards. An ant has a start state and one or more termination conditions. In a single step the ant moves from a current vertex v of the construction graph to one of its successors. This move is chosen due to a probabilistic rule and depends on the pheromone values on the edges, heuristic information associated with components and connections in the neighborhood of v, the ant's private memory, and the problem constraints. When adding a component to the solution the ant builds up, it may update the pheromone value of the connection that corresponds to this solution. This is not always done. Usually the pheromone values are updated after the complete solution has been build. Here the ant retraces the path it has gone to build up the solution and increases the pheromone values along these edges.

Let $C = (V, E)$ be the construction graph of a given problem. The pheromone value of an edge $e = (u, v) \in E$ is denoted by $\tau_{(u,v)}$. In addition it is possible to assign to each edge $(u, v) \in E$ a heuristic information $\eta_{(u,v)}$. We assume that an ant is at vertex u and denote the set of allowed successors by $N(u)$. Due to the problem constraints this set may be a subset of the successors of u in C. The probability that the ant visits the vertex $v \in N(u)$ in the next step is given by

$$p_v = \frac{[\tau_{(u,v)}]^\alpha \cdot [\eta_{(u,v)}]^\beta}{\sum_{w \in N(v)} [\tau_{(u,w)}]^\alpha \cdot [\eta_{(u,w)}]^\beta}.$$

Here $\alpha, \beta \geq 0$ are parameters that determine the importance of the pheromone values respectively the heuristic information. In the algorithm we consider for the analysis we will use $\alpha = 1$ and $\beta = 0$. This means that no heuristic information is taken into account and that the probability of choosing a specific vertex $v \in N(u)$ is proportional to the values of all edges going from u to vertices of $N(u)$.

In the update procedure of an ACO algorithm, the pheromone values are usually decreased by an amount that depends on the value before the update and the evaporation factor ρ, $0 \leq \rho \leq 1$. Let $\tau_{(u,v)}$ be the pheromone value on edge $(u, v) \in E$ before the update. The value is decreased to $(1 - \rho)\tau_{(u,v)}$ in a first step. This implies that information about which paths are gone so far gets lost during the run of the algorithm and helps to escape from local optima. In addition the pheromone values on edges an ant a_i has traversed are increased by a value Δ_i that may depend on ρ as well as on the function value of the solution the ant a_i has constructed. Hence, the pheromone value $\tau'_{(u,v)}$ of edge (u, v) after the update is given by

$$\tau'_{(u,v)} = (1 - \rho)\tau_{(u,v)} + \sum_{i=1}^{k} \Delta_i.$$

There are different possibilities which ants to take into account for the update. If all ants of the colony leave pheromone values on the edges this is known as the AS-update rule. This is the update rule of the Ant System (AS) which was the first ACO algorithm proposed in the literature (see Dorigo, Maniezzo, and Colorni (1991)). Using the AS-update the amount by which an ant increases a pheromone value should depend on the function value of the constructed solution as otherwise the pheromone values are totally independent of the function f that should be optimized. Therefore, it would not be possible to direct the search. In the case of the IB-update rule, where IB stands for iteration best, the ants that have constructed the best solutions of the last iteration update the pheromone values along the edges they have taken. Such an update introduces a much stronger bias towards the best solutions found so far. In the case of the best so far update, BS-update for short, the pheromone values on the edges are only increased if there has been a solution constructed in the last iteration that is at least as good as the best solution constructed since the first iteration of the algorithm. For our analysis in Chapter 8 we will consider a simple ACO algorithm that uses the BS-update rule.

3.3 Other randomized search heuristics

In this section we describe other import randomized heuristics that have been proposed. One important method is randomized local search (RLS). This can be seen as a simplification of the perhaps simplest evolutionary algorithms called (1+1) EA. In the case of a runtime analysis for the (1+1) EA, RLS is often considered in a first step and the results are later adjusted to the EA. Local search procedures work with a predefined neighborhood and have problems if there is no better solution in this neighborhood than the current one. Then they get stuck in local optima. To escape from local optima the Metropolis algorithm (MA) allows to accept worsenings with a certain probability that depends on a parameter that is called the temperature. It has been shown to be useful in an approach called simulated annealing (SA) to vary this temperature over the time starting with a high temperature and cooling it down during the run of the algorithm.

3.3.1 Randomized local search

Apart from sampling in each iteration a search point from the given search space uniformly at random, randomized local search seems to be the simplest randomized search heuristic that can be considered. RLS works in each iteration with one single solution s. A new solution s' is constructed from s by choosing one individual from the neighborhood of s. s is replaced by s' if s' is not inferior to s. The definition of the neighborhood is a crucial parameter. If it is too small RLS often gets stuck in local optima. If the neighborhood is too large even individuals that are close to the current solution may only get a too small probability of being chosen in the next step and RLS behaves like random sampling search points from the search space independently of s. Considering problems from the search space $\{0,1\}^n$ RLS often uses a neighborhood that is defined by all search points that have Hamming distance 1 or 2 to the current solution s.

In the case of local search procedures the complexity class PLS (polynomial local search) has been introduced by Johnson, Papadimitriou, and Yannakakis (1988). Some important problems that are PLS-complete can be found in Papadimitriou, Schäffer, and Yannakakis (1990). For each instance I of a given problem P we have a set of feasible solutions F_I, such that given I and F_I, it is easy to decide for a solution s whether $s \in F_I$ holds. Then we can produce in polynomial time a feasible solution $s_0 \in F_I$, which is usually called the initial solution of a local search heuristic. Evaluating s with respect to the fitness function f should be possible in polynomial time. In addition the test whether s is local optimal should be possible in polynomial time. If there is a better s' in the neighborhood of s such a solution is produced. We denote by A the following computational problem: Given an input I, find a locally optimal solution $s \in F_I$. A PLS-reduction from problem A to another problem B is defined in terms of two polynomial computable functions g and h. Given an instance I of A, g computes an instance $g(I)$ of B such that for any local optimum s of $g(I)$, $h(s, I)$ is a local optimum of I. The functions g and h should not only be polynomial, but also logspace-computable.

The first problem that has been shown to be PLS-complete is FLIP. In this problem a circuit with many binary inputs and outputs is given. A solution is an input to the circuit and the fitness is the output read as a binary integer. The neighborhood of an input is defined by all solutions s' that have Hamming distance 1 to the current solution s. In the case that we are not only interested in finding a local optimal solution, but a local optimum reachable by local improvements from a given feasible solution, FLIP and many PLS-complete problems (including the following two problems) become PSPACE-complete. Problems defined in this way are called the standard local optimum version of the given problem. 2SAT is PLS-complete with a neighborhood defined by solutions of Hamming distance 1. Here a Boolean formula in CNF with two literals per clause is given. In addition weights are assigned to the clauses. The aim is to maximize the sum of weights of clauses that are satisfied. In the case of the MAXCUT problem a weighted graph is given. A solution is a set of nodes and the aim is to maximize the sum of weights of edges leaving this set of nodes. In the case of the SWAP-neighborhood, where the symmetric difference of two solutions contains exactly one node, the problem is known to be PLS-complete.

One of the best-known examples for local search is the Lin-Kernighan algorithm (see Lin and Kernighan (1973)) for the traveling salesperson problem (TSP). In the traveling salesperson problem a complete graph with positive edge weights is given and one searches for a tour of minimum cost which visits all nodes exactly once and returns to the start node. The Lin-Kernighan algorithm relies on the 2-exchange neighborhood for the TSP. A tour T' is a neighbor of a tour T if T and T' differ exactly by 2 edges. The Lin-Kernighan algorithm allows changes of arbitrarily many edges. It has been shown that a variant of the Lin-Kernighan algorithm is PLS-complete and that the standard local optimum version of the problem is PSPACE-complete.

Despite these negative results local search is a technique that has been used in many different applications and produced good results. Papadimitriou, Schäffer, and Yannakakis (1990) state that there are many empirically fast algorithms for the mentioned problems and remark that there is no family of PSPACE-complete problems that behaves empirically in such a positive way.

Despite its simplicity RLS is not easy to analyze. RLS sampling in each iteration solutions defined by sampling solutions that have Hamming distance 1 or 2 from the current solution s has been considered by Giel and Wegener (2003) for the maximum matching problem as a starting point for the analysis of a simple evolutionary algorithm called (1+1) EA. They have shown that RLS finds a maximum matching for graphs that consist of one single path with m edges in expected time $O(m^4)$. In the case that the input graph is a tree the expected time of RLS to obtain an optimal solution is $O(m^6)$ which has been shown by Giel and Wegener (2004) in an extension of their previous work.

Another important analysis for RLS has been done by Witt (2005). He has considered an NP-hard scheduling problem with n jobs on two identical machines and shown that RLS finds a factor 4/3-approximation of an optimal solution in expected time $O(n^2)$. By presenting an example instance he has shown the RLS can not achieve a better approximation ratio in expected polynomial time as there is a constant probability that RLS needs an exponential optimization time. As randomized search heuristics often use multi-starts to cope with small failure probabilities, he has investigated whether multi-starts can help. His analysis shows that multi-starts of RLS lead to a polynomial-time randomized approximation scheme (PRAS). In addition he has analyzed RLS on random instances of the partition problem. Witt also uses RLS as a starting point for the analysis of the (1+1) EA on the partition problem and the mentioned results transfer to the (1+1) EA.

3.3.2 Metropolis algorithm

In contrast to RLS the following two approaches accept worsenings during the optimization process. The acceptance of a worsening depends on the difference of the fitness values of s and s' and a so-called temperature T. In the case of the Metropolis algorithm (MA) this temperature is a fixed parameter and therefore constant during the optimization process. We assume that we are considering a function f that should be maximized. In the case that $f(s') \geq f(s)$ holds, s is replaced by s'. In the other case s is replaced by s' with probability $M(s, s', T) = e^{-\frac{f(s)-f(s')}{T}}$ where M is called the Metropolis function.

MA has been subject to the rigorous analysis with respect to its runtime for the NP-hard graph bisection problem (Jerrum and Sorkin (1998)). Let $G = (V, E)$ be an undirected graph where $|V|$ is even. A bisection of G is a partitioning of V into sets L and R with $|L| = |R| = n/2$. The cut-width of a bisection is defined as the number of edges that have exactly one endpoint in L and one endpoint in R. One is interested in finding a bisection with minimum cut-width. Jerrum and Sorkin have considered MA for finding an optimal bisection of a random graph $G = (V, E)$ where an edge between vertices of the same partition occurs with probability p and an edge between vertices of L and R occurs with probability r. In the case that $p - r = \Theta(n^{\Delta-2})$ for a parameter Δ with $3/2 < \Delta \leq 2$, such a random graph specifies with high probability a planted bisection of density r which separates L and R that have a slightly higher density p (see Bui, Chaudhuri, Leighton, and Sipser (1984)). Then it can be shown that MA for an appropriate choice of T finds the optimal solution in about $O(n^2)$ steps with high probability if $\Delta \geq 11/6$.

3.3.3 Simulated annealing

Simulated annealing (SA) can be seen as MA that uses different temperatures during the run of the algorithm. Starting with a temperature T_0 the temperature is decreased during the optimization process according to a cooling schedule. Such a cooling schedule can be adaptive or non-adaptive. In the case of a non-adaptive cooling schedule the temperature T_i is known in advance for all time steps i. In the case of adaptive cooling schedules

Figure 3.1: Connected triangles with two different weight profiles

the temperature for a given time step i may depend on the history of sampled search points.

For a long time there were only artificial example functions (Sorkin (1991)) where it could be proven that a cooling schedule can be useful to reduce the runtime significantly. Recently, Wegener (2005b) has presented the first "natural" example where this is the case. He has shown that SA can outperform MA for each fixed temperature on a class of instances of the minimum spanning tree problem. Wegener has investigated connected triangles (see Figure 3.1) with $m = 6n$ edges and $4n + 1$ vertices. The structure of this graph is the same as the triangle part of the graph we will investigate in Chapter 7 for the analysis of evolutionary algorithms until they have computed a minimum spanning tree. The number of triangles equals $2n$. Each triangle gets a weight profile (w_1, w_2, w_3) which is the ordered vector of the three edge weights. The basic idea is to construct weight profiles such that for each fixed temperature it is hard to optimize all triangles while an appropriate cooling scheduling is able to optimize all triangles. Wegener uses n triangles with the weight profile $(1, 1, m)$ and n triangles with the weight profile (m^2, m^2, m^3). Then he distinguishes between high temperatures ($T \geq m$) and low temperatures ($T < m$). He shows that high temperatures are not able to optimize the triangles with the weight profile $(1, 1, m)$ and low temperatures are not able to optimize the triangles with weight profile (m^2, m^2, m^3) in a polynomial number of steps. Hence, different temperatures are necessary to find an optimal solution solution quickly. An optimal solution can be obtained in a polynomial number of steps by using an appropriate cooling scheduling in SA.

Part II

Algorithms and Basic Methods for the Analysis

Chapter 4

Algorithms to be analyzed

In this chapter we introduce the randomized search heuristics that will be subject to the analysis throughout this work. We start by describing algorithms for single-objective optimization problems in Section 4.1. There we consider different variants of RLS and variants of a well-known evolutionary algorithm called (1+1) EA. After that we introduce a simple ACO algorithm where in each iteration one ant performs a random walk on the construction graph. In Section 4.2 we take a look at the multi-objective randomized search heuristics that will be analyzed. These heuristics can be seen as a generalization of RLS and the (1+1) EA to the multi-objective case.

4.1 Single-objective optimization problems

In this section we want to introduce the randomized search heuristics that we will consider for single-objective optimization problems. We investigate heuristics for discrete search spaces. Most of the problems we examine in this work are graph problems where one searches for a good set of edges or a good permutation of the edges. For a given graph with n vertices and m edges the dimension of the search space is m. We consider three variants of randomized local search. One is working with bitstrings of length m and two are working with permutations of m elements which distinguish from each other by the chosen operator to produce new solutions. After having defined these algorithms we extend them to evolutionary algorithms. In the case of ACO algorithms we consider a simple algorithm where one ant produces solutions that are bitstrings. We will see later that this algorithm can be seen as a generalization of the (1+1) EA when considering the binary case.

4.1.1 Randomized local search

Randomized local search in the binary case produces from a current solution $s \in \{0,1\}^m$ a new one s' using the following operator:

- Choose $i \in \{1, \ldots, m\}$ randomly and flip the ith bit of s.

Here we use the notion "choose randomly" for a choice according to the uniform distribution.

We will sometimes also call the operator of RLS that produces new solutions mutation operator. Flipping one single bit is not useful for most graph problems. Often the number of ones (or edges) is the same for all good search points, e. g. for TSP or minimum spanning trees. Then all Hamming neighbors of good search points are bad implying that we have many local optima. Therefore, we work with the larger neighborhood of Hamming distance 2. This mutation operator has already been discussed for maximum matchings by Giel and Wegener (2003).

We will investigate the runtime of randomized local search for spanning tree problems and consider the following variant of RLS for the binary case, which we describe for minimizing a fitness function f.

Algorithm 4.1.1 ($\mathrm{RLS_b}$)

1. *Choose* $s \in \{0,1\}^m$ *randomly.*
2. *Choose* $b \in \{0,1\}$ *randomly. If* $b = 0$*, choose* $i \in \{1,\ldots,m\}$ *randomly and define* s' *by flipping the* i*th bit of* s*. If* $b = 1$*, choose* $(i,j) \in \{(k,l) \mid 1 \le k < l \le m\}$ *randomly and define* s' *by flipping the* i*th and the* j*th bit of* s*.*
3. *Replace* s *by* s' *if* $f(s') \le f(s)$*.*
4. *Repeat Steps 2 and 3 forever.*

For all the randomized search heuristics we consider no stopping criterion is defined. In applications this is, of course, necessary. Often such an algorithm is stopped after a predefined number of iterations or if no progress has been made for a certain number of steps. We consider the algorithms we analyze as infinite stochastic processes and are interested in the number of fitness evaluations until a given task has been achieved. In the case of exact optimization the number of fitness evaluations until an optimal solution has been produced is investigated. Often the expectation of this value is analyzed and called the expected optimization time of the considered algorithm. Especially in the case where one can not hope to compute optimal solutions in a polynomial number of steps, e.g. for NP-hard problems, one is interested in the number of fitness evaluations until the algorithm has produced a good approximation of an optimal solution.

In the case that we are searching for a good permutation of the input elements, jumps and exchanges (see Section 3.1) are popular operators that lead to new solutions. Both operators have been integrated into one mutation operator by Scharnow, Tinnefeld, and Wegener (2002, 2004) for the sorting problem. We consider these two operators separately from each other and introduce two algorithms that search for good permutations of given input elements. The algorithm RLS_p executes in one mutation step exactly one jump

operation. This jump is chosen according to the uniform distribution among all possible jumps which means that the positions i and j are chosen uniformly at random from the set $\{1, \ldots, m\}$. Our algorithm starts with a permutation π which is chosen randomly from the set S_m that consists of all permutations of m elements. We will analyze such randomized search heuristics until they have found a good permutation of the edges of a given graph for the Eulerian cycle problem. The underlying fitness function should be maximized. Therefore, we describe RLS_p as follows.

Algorithm 4.1.2 (RLS_p)

1. *Choose $\pi \in S_m$ randomly.*
2. *Choose i und j randomly and define π' by executing $jump(i, j)$ on π.*
3. *Replace π by π' if $f(\pi') \geq f(\pi)$.*
4. *Repeat Steps 2 and 3 forever.*

In a similar way we define the algorithm RLS_p^* which uses exchange operations instead of jumps. As already discussed in Section 3.1 exchange operations do not seem to be as flexible as jump operations. We will see later that this can make the difference between a polynomial and an infinite expected optimization time for randomized local search.

Algorithm 4.1.3 (RLS_p^*)

1. *Choose $\pi \in S_m$ randomly.*
2. *Choose i und j randomly and define π' by executing $exchange(i, j)$ on π.*
3. *Replace π by π' if $f(\pi') \geq f(\pi)$.*
4. *Repeat Steps 2 and 3 forever.*

4.1.2 Simple evolutionary algorithms

The evolutionary algorithms that we consider for single-objective optimization problems use a population of size one and produce at each time step one single child. They can be seen as variants of RLS which we have introduced in the last section with a more flexible mutation operator. Usually, a mutation operator in this scenario should be able to search globally. Here each search point of the considered search space should get a positive probability of being chosen in the next step. Again we consider the algorithm for the search space $\{0, 1\}^m$ first. The perhaps simplest evolutionary algorithm that can be considered in this case is (1+1) EA_b. Starting with a randomly chosen bitstring s of length m the algorithm produces in each iteration a child by flipping each bit of s with probability $1/m$. We can describe (1+1) EA_b for minimizing a fitness function f as follows.

Algorithm 4.1.4 ((1+1) EA$_b$)

1. *Choose $s \in \{0,1\}^m$ randomly.*
2. *Produce s' by flipping each bit of s independently of the other bits with probability $1/m$.*
3. *Replace s by s' if $f(s') \leq f(s)$.*
4. *Repeat Steps 2 and 3 forever.*

(1+1) EA$_b$ has been the subject of the first analyses of evolutionary algorithms with respect to their expected optimization time. In the beginning the behavior of this algorithm on pseudo-boolean functions that depend on n variables has been considered. Some of first main results have been obtained by Droste, Jansen, and Wegener (2002). It has been shown that the expected time to reach an optimal search point by this algorithm in the considered search space is always bounded above by n^n as the probability to choose an optimal search point in the next step is at least n^{-n}. More detailed analyses consider pseudo-boolean functions with different properties. One major result is that the expected optimization time on linear functions is $O(n \log n)$. The class of functions of degree two is too huge to get a polynomial upper bound on the runtime for each function as optimizing polynomials of degree at least two is NP-hard.

Similar to (1+1) EA$_b$ we define the EAs for permutation problems. For large m the binomial distribution can be approximated very well by the Poisson distribution. In addition we assure that at least one operation is executed in each mutation step. We consider the EAs that search for good permutations with respect to the Eulerian cycle problem and define them for maximizing a fitness function f. The algorithm (1+1) EA$_p$ which uses jumps for mutation can be described as follows.

Algorithm 4.1.5 ((1+1) EA$_p$)

1. *Choose $\pi \in S_m$ randomly.*
2. *Define π' in the following way. Choose l according to a Poisson distribution with parameter $\lambda = 1$ and perform sequentially $l+1$ randomly chosen jump operations to produce π' from π.*
3. *Replace π by π' if $f(\pi') \geq f(\pi)$.*
4. *Repeat Steps 2 and 3 forever.*

For RLS$_p$ and RLS$_p^*$ we will show that switching from jumps to exchanges can turn a polynomial expected optimization time into an infinite one. In the case of the EAs we consider, each search point has a positive probability of being chosen in the next step. Nevertheless we will show in Chapter 6 that switching from jumps to exchanges can make

the difference between a polynomial and an exponential expected optimization also for simple EAs. We define the algorithm (1+1) EA^*_p which uses exchanges instead of jumps similar to the previous one.

Algorithm 4.1.6 ((1+1) EA^*_p)

1. *Choose $\pi \in S_m$ randomly.*
2. *Define π' in the following way. Choose l according to a Poisson distribution with parameter $\lambda = 1$ and perform sequentially $l+1$ randomly chosen exchange operations to produce π' from π.*
3. *Replace π by π' if $f(\pi') \geq f(\pi)$.*
4. *Repeat Steps 2 and 3 forever.*

4.1.3 A simple ant colony optimization algorithm

Gutjahr (2003) has considered a graph-based ant system and investigated under which conditions such an algorithm converges to an optimal solution. We consider a simple graph-based ant system metaheuristic that has been inspired by this algorithm. Such a heuristic produces solutions by random walks on a construction graph. Let $C = (V, E)$ be the directed construction graph with a designated start vertex s and pheromone values τ on the edges. Starting at s, an ant traverses the construction graph depending on the pheromone values using Algorithm 4.1.7. Assuming that the ant is at vertex v, it moves to a successor w of v, where w is chosen proportional to the pheromone values of all non-visited successors of v. The process is iterated until a situation is reached where all successors of the current vertex v have been visited.

Algorithm 4.1.7 (Construct(C, τ))

1. *v:=s, mark v as visited.*
2. *While there is a successor of v in C that has not been visited:*
   (a) *Let N_v be the set of non-visited successors of v and $T := \sum_{w \in N_v} \tau_{(v,w)}$.*
   (b) *Choose one successor w of v where the probability of selection of any fixed $u \in N_v$ is $\tau_{(v,u)}/T$.*
   (c) *Mark w as visited, set $v := w$ and go to 2..*
3. *Return the solution x and the path $P(x)$ constructed by this procedure.*

Based on this construction procedure, solutions of our simple ACO algorithm (see Algorithm 4.1.8) called 1-ANT are constructed. In the initialization step, each edge gets a pheromone value of $1/|E|$ such that the pheromone values sum up to 1. After that, an initial

solution x^* is produced by a random walk on the construction graph and the pheromone values are updated with respected to this walk. In each iteration, a new solution x is constructed and the pheromone values are updated if this solution is not inferior to the currently best solution x^*. We formulate our algorithm for maximization problems although it can be easily adapted to minimization.

Algorithm 4.1.8 (1-ANT)

1. *Set* $\tau_{(u,v)} = 1/|E|$ *for all* $(u,v) \in E$.
2. *Compute* x *(and* $P(x)$*) using Construct*(C, τ).
3. *Update*$(\tau, P(x))$ *and set* $x^* := x$.
4. *Compute* x *(and* $P(x)$*) using Construct*(C, τ).
5. *If* $f(x) \geq f(x^*)$, *Update*$(\tau, P(x))$ *and set* $x^* := x$.
6. *Go to 4.*

Again we are interested in the optimization time. In this case it equals the number of constructed solutions until the algorithm has produced an optimal search point.

We take a general view and consider optimization for pseudo-boolean goal functions $f\colon \{0,1\}^n \to \mathbb{R}$ for $n \geq 3$. We investigate the construction graph $C_{\text{bool}} = (V, E)$ (see Figure 4.1) with $s = v_0$. In the literature this graph is known as *Chain* (Gutjahr (2006)). Optimizing bitstrings of length n, the graph has $3n + 1$ vertices and $4n$ edges. The decision whether a bit x_i, $1 \leq i \leq n$, is set to 1 is made at node $v_{3(i-1)}$. In case that the edge $(v_{3(i-1)}, v_{3(i-1)+1})$ is chosen, x_i is set to 1 in the constructed solution. Otherwise $x_i = 0$ holds. After this decision has been made, there is only one single edge which can be traversed in the next step. In case that $(v_{3(i-1)}, v_{3(i-1)+1})$ has been chosen, the next edge is $(v_{3(i-1)+1}, v_{3i})$, and otherwise the edge $(v_{3(i-1)+2}, v_{3i})$ will be traversed. Hence, these edges have no influence on the constructed solution and we can assume $\tau_{(v_{3(i-1)}, v_{3(i-1)+1})} = \tau_{(v_{3(i-1)+1}, v_{3i})}$ and $\tau_{(v_{3(i-1)}, v_{3(i-1)+2})} = \tau_{(v_{3(i-1)+2}, v_{3i})}$ for $1 \leq i \leq n$. We call the edges $(v_{3(i-1)}, v_{3(i-1)+1})$ and $(v_{3(i-1)+1}, v_{3i})$ 1-*edges* and the other edges 0-*edges*. The edges $(v_{3(i-1)}, v_{3(i-1)+1})$ and $(v_{3(i-1)}, v_{3(i-1)+2})$ as well as $(v_{3(i-1)+1}, v_{3i})$ and $(v_{3(i-1)+2}, v_{3i})$ are called *complementary* to each other.

The pheromone values are chosen such that at each time $\sum_{(u,v)\in E} \tau_{(u,v)} = 1$ holds. In addition, it seems to be useful to have bounds on the pheromone values (see e. g. Dorigo and Blum (2005)) to ensure that each search point has a positive probability of being chosen in the next step. We restrict each $\tau_{(u,v)}$ to the interval $\left[\frac{1}{2n^2}, \frac{n-1}{2n^2}\right]$ and ensure $\sum_{(u,\cdot)\in E} \tau_{(u,\cdot)} = \frac{1}{2n}$ for $u = v_{3i}$, $0 \leq i \leq n-1$, and $\sum_{(\cdot,v)} \tau_{(\cdot,v)} = \frac{1}{2n}$ for $v = v_{3i}$, $1 \leq i \leq n$. This can be achieved by normalizing the pheromone values after an update and replacing the current value by $\frac{1}{2n^2}$ if $\tau_{(u,v)} < \frac{1}{2n^2}$ and by $\frac{n-1}{2n^2}$ if $\tau_{(u,v)} > \frac{n-1}{2n^2}$ holds. Depending on whether the edge (u,v) is

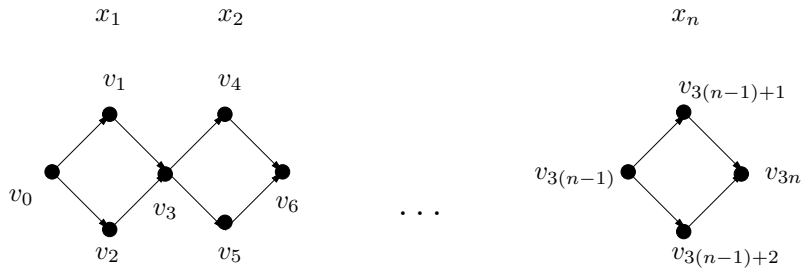

Figure 4.1: Construction graph C_{bool} for pseudo-boolean optimization

contained in the path $P(x)$ of the accepted solution x, the pheromone values are updated to τ' in the procedure Update$(\tau, P(x))$ as follows:

$$\tau'_{(u,v)} = \min\left\{\frac{(1-\rho)\cdot\tau_{(u,v)}+\rho}{1-\rho+2n\rho}, \frac{n-1}{2n^2}\right\} \quad \text{if } (u,v) \in P(x)$$

and

$$\tau'_{(u,v)} = \max\left\{\frac{(1-\rho)\cdot\tau_{(u,v)}}{1-\rho+2n\rho}, \frac{1}{2n^2}\right\} \quad \text{if } (u,v) \notin P(x).$$

Due to the bounds on the pheromone values, the probability of fixing x_i as in an optimal solution is at least $1/n$. Hence, 1-ANT finds an optimum for each pseudo-boolean function $f$regardless of ρ in expected time at most n^n.

4.2 Multi-objective optimization problems

The rigorous analysis of the expected optimization time of evolutionary algorithms is not easy. Most of such results are on simple evolutionary algorithms like the (1+1) EA_b. This is even more true for multi-objective optimization. Therefore, we investigate and analyze a simple algorithm called SEMO (Simple Evolutionary Multi-Objective Optimizer) due to Laumanns, Thiele, Zitzler, Welzl, and Deb (2002).

The fitness of a search point s is given by a vector $f(s) = (f_1(s), \ldots, f_k(s))$. W. l. o. g. we assume that each function f_i should be minimized and write $f(s) \leq f(s')$ iff $f_i(s) \leq f_i(s')$ holds for all i, $1 \leq i \leq k$. A solution s domintates a solution s' iff $f(s) \leq f(s')$ and $f(s) \neq f(s')$ holds. If s dominates s' we also say that $f(s)$ dominates $f(s')$. We will analyze the multi-objective evolutionary algorithms (MOEAs) until they have achieved certain goals for spanning tree problems. In the case of polynomially solvable problems we are interested in the time until for each Pareto optimal objective vector a solution has been produced whereas in the case of NP-hard problems we are interested in the time to achieve a good approximation of the Pareto front. We search for a set of edges that is described by a bitstring of length m. The algorithm starts with an initial solution $s \in \{0,1\}^m$. All non-dominated solutions are stored in the population P. In each step a search point from P is chosen uniformly at random and one bit is flipped to obtain a new search point s'.

The new population contains for each non-dominated fitness vector $f(s)$, $s \in P \cup \{s'\}$, one corresponding search point and in the case that $f(s')$ is not dominated s' is chosen.

In the case of multi-objective optimization we are interested in the number of rounds until a desired goal has been achieved. In the case where we consider exact optimization, we are interested in the number of rounds until $f(P) := \{f(s)|s \in P\}$ equals the Pareto front. As even simple single-objective problems get NP-hard when an additional objective has to be optimized at the same time, we are also interested in approximating the Pareto front for difficult problems.

Algorithm 4.2.1 (SEMO)

1. *Choose an initial solution s.*
2. *Determine $f(s)$ and initialize $P := \{s\}$.*
3. *Repeat*
   (a) *Choose $s \in P$ randomly.*
   (b) *Choose $i \in \{1, \ldots, m\}$ randomly.*
   (c) *Define s' by flipping the ith bit of s.*
   (d) *Determine $f(s')$,*
   (e) *Let P unchanged, if there is an $s'' \in P$ such that $f(s'') \leq f(s')$ and $f(s'') \neq f(s')$*
   (f) *Otherwise, exclude all s'' where $f(s') \leq f(s'')$ from P and add s' to P.*

Note that the described algorithm differs from the original version of SEMO by replacing an individual s'' of P by s' if $f(s'') = f(s')$ holds. Applying our version of SEMO to a single-objective optimization problem, we obtain RLS where in each step a single bit is flipped. All results obtained in this work also hold for the original version of SEMO but it seems to be more typical for search heuristics to replace search points by other ones with the same quality (e.g., simulated annealing works this way). If SEMO starts with a search point s that is a local optimum, then $P = \{s\}$ forever.

As in the case of RLS_b and (1+1) EA_b this local mutation operator is motivated by the fact that this choice simplifies the analysis. Giel (2003) has considered an algorithm called Global SEMO (GSEMO) which uses the usual mutation operator of evolutionary algorithms.

Algorithm 4.2.2 (Global SEMO (GSEMO))

1. *Choose an initial solution s.*
2. *Determine $f(s)$ and initialize $P := \{s\}$.*
3. *Repeat*
   (a) *Choose $s \in P$ randomly.*
   (b) *Define s' by flipping each bit of s independently of the other bits with probability $1/m$.*
   (c) *Determine $f(s')$,*
   (d) *Let P unchanged, if there is an $s'' \in P$ such that $f(s'') \leq f(s')$ and $f(s'') \neq f(s')$*
   (e) *Otherwise, exclude all s'' where $f(s') \leq f(s'')$ from P and add s' to P.*

Global SEMO applied to single-objective optimization problems equals (1+1) EA_b if the initial solution is chosen uniformly at random. In Chapter 10 we will compare SEMO and GSEMO with RLS_b and (1+1) EA_b for the computation of minimum spanning trees.

Chapter 5

Basic Methods for the Analysis

Until the early '90s of the last century theory on evolutionary algorithms has mainly been dealt with the consideration of convergence of EAs or results that show the behavior of an EA in one single iteration. The first runtime analysis of an EA has been given by Mühlenbein (1992). Evolutionary algorithms are randomized search heuristics but for a long time they have not been analyzed in the way people from the classical algorithm community would normally do. The main reason for that is that the people that have worked on theoretical aspects of evolutionary computation have had another background than people from theoretical computer science or discrete mathematics have. Regarding evolutionary algorithms as a class of randomized algorithms a lot of strong methods are available. Such methods have already been applied in the field of randomized algorithms (see e. g. Motwani and Raghavan (1995)). A very important issue when analyzing the runtime of EAs is the application of large deviation inequalities such as Chernoff bounds or Markov's inequality. Another useful method is to follow the considerations for the coupon collectors problem. Since the mid '90s, a lot of new methods for analyzing the runtime of EAs have been obtained. In this chapter, we want to discuss some important methods that have been used. These methods will be applied in our analysis of evolutionary algorithms for combinatorial optimization problems.

To show how to apply different methods that have been developed, we consider the class of linear pseudo-boolean functions. A linear pseudo-boolean function $f\colon \{0,1\}^n \to \mathbb{R}$ is defined by

$$f(x) = w_1x_2 + w_2x_2 + \ldots + w_nx_n,$$

where $w_i \in \mathbb{Z}$.

W. l. o. g. we assume that all w_i attain non-negative values. The case of (partially) negative weights can be handled analogously to the following investigations as a weight $w_i \neq 0$ determines independently of the other weights whether the bit x_i has to be set to 1 or 0 in an optimal solution. In the case that some weights are 0 the function value does not depend on the corresponding bits. The upper bound given in Theorem 5.3.1 also holds

in this case, but the lower bound given in Theorem 5.2.1 needs the condition that there are $\Theta(n)$ weights which are distinct from 0.

5.1 Fitness-based partitions

This simple method has been used for a wide class of problems. We assume that we are considering a randomized search heuristic that works in each iteration with one solution that produces one offspring. All variants of RLS and the (1+1) EA we have discussed in Section 4.1 fit into this scenario. Assume that we are working in a search space S and consider w.l.o.g. a function $f\colon S \to \mathbb{R}$ that should be maximized. S is partitioned into disjoint sets $A_1, \ldots, A_m$ such that $A_1 <_f A_2 <_f \ldots <_f A_m$ holds, where $A_i <_f A_j$ means that $f(a) < f(b)$ holds for all $a \in A_i$ and all $b \in A_j$. In addition A_m contains only optimal search points. We denote for a search point $x \in A_i$ by $p(x)$ the probability that in the next step a solution $x' \in A_{i+1} \cup \ldots \cup A_m$ is produced. Let $p_i = \min_{a \in A_i} p(x)$ be the smallest probability of producing a solution with a higher partition number.

Lemma 5.1.1 *The expected optimization time of a randomized search heuristic that works at each time with a population of size* 1 *and produces at each time step a new solution from the current solution is upper bounded by* $\sum_{i=1}^{m-1}(1/p_i)$.

Proof: The expected time of a success for independent Bernoulli trials with probability p is $1/p$. Hence the expected time to produce from a search point $x \in A_i$ a search point x' with $x' \in A_j$, $j > i$, is upper bounded by $1/p_i$. This implies that the expected time until an optimal search point has been produced is upper bounded by $\sum_{i=1}^{m-1}(1/p_i)$. □

To come up with good upper bounds using this method one has to use a good partitioning of the search space such that there are not too many partitions and that there is a high probability of leaving the current partition and producing a search point in a better one.

We consider a simple example. OneMax: $\{0,1\}^n \to \mathbb{R}$ is a simple linear pseudo-boolean function where $w_i = 1$, $1 \leq i \leq n$, holds. It is defined by $\text{OneMax}(x) = \sum_{i=1}^{n} x_i$ and should be maximized. The function returns for a bitstring x of length n the number of ones in x. We consider (1+1) EA_b for maximization problems, where a new solution is accepted if its fitness value is not smaller than the value of the up to now best solution.

Theorem 5.1.2 *The expected optimization time of (1+1)* EA_b *on OneMax is* $O(n \log n)$.

Proof: The search space is partitioned into $n+1$ sets $A_0, \ldots, A_n$ where A_i contains all solutions x with $\text{OneMax}(x) = \text{i}$. Assume that the currently best solution x belongs to A_{n-k}. Then there are exactly k 0-bits that can be flipped to obtain an improvement. The probability for an improvement in the next step is at least $\frac{k}{n}(1-\frac{1}{n})^{n-1} \geq \frac{k}{en}$. Hence, the

expected waiting time for an improvement is upper bounded by en/k. Summing up the waiting times for the different values of k we get

$$\sum_{k=1}^{n} \frac{en}{k} = en \cdot \sum_{k=1}^{n} \frac{1}{k} = O(n \log n). \qquad \square$$

In the case that one works with a larger population often an individual with the highest partition number in the population is considered. Then one can analyze the time until this individual has become an optimal one. The method works nearly the same as in the case of a population of size 1 but one often has to add an additional factor to choose the right individual in the next step. We will apply the method of fitness-based partitions in this way for multi-objective problems where we have to consider the population size. The MOEAs introduced in Section 4.2 have a population that is determined by the number of solutions that do not dominate each other and have different fitness vectors.

5.2 Chernoff bounds and coupon collectors

Large deviation inequalities have widely been used in the analysis of randomized algorithms. In the case of randomized search heuristics they are often useful to show the typical behavior of such a heuristic. We consider (1+1) EA_b which chooses the initial solution x uniformly at random from $\{0,1\}^n$ by setting each bit with equal probability to 0 or 1. Hence, n Bernoulli trials are considered where $\text{Prob}(x_i = 1) = \text{Prob}(x_i = 0) = 1/2$, $1 \leq i \leq n$, holds. The expected number of ones in the initial solution is therefore $n/2$ and there are at most $2n/3$ ones in the initial bitstring with probability $1 - e^{-\Omega(n)}$ using Chernoff bounds (see Appendix A.2.2).

In the coupon collector's problem (see e. g. Motwani and Raghavan (1995)) n different coupons are given and at each time step a coupon is chosen uniformly at random among all coupons. Let t be the number of trials. Then one studies the number of trials until each of the n coupons has been chosen at least once. The expected number of trials until each coupon has been chosen at least once is $\Theta(n \log n)$ (see Appendix A.3.4). Using Chernoff bounds and the ideas of the coupon collectors problem, it is easy to obtain a lower bound of $\Omega(n \log n)$ on each linear pseudo-boolean function with non-zero weights. To show how to use Chernoff bounds and the ideas of the coupon collectors problem we present the proof which can be found in Droste, Jansen, and Wegener (2002). W. l. o. g. we assume that all weights attain positive values. Hence, the only optimal solution is the bitstring $(1, \ldots, 1)$.

Theorem 5.2.1 *The expected optimization time of (1+1) EA_b on each linear pseudo-boolean function with non-zero weights is $\Omega(n \log n)$.*

Proof: Using Chernoff bounds the expected number of zeros in the initial bitstring is at least $n/3$ with probability $1 - e^{-\Omega(n)}$. To obtain the proposed lower bound, we analyze the

expected time until each of the 0-bits has been flipped at least once under the condition that there are at least $n/3$ 0-bits after initialization. This is done in a similar fashion as in the case of the coupon collector's theorem.

Let t be a specific number of steps. The probability that a specific 0-bit has not been flipped at least once in t steps is $(1-1/n)^t$. Hence, the probability that it has flipped at least once in t steps is $1-(1-1/n)^t$ and the probability that each of the $n/3$ 0-bits has flipped at least once is $(1-(1-1/n)^t)^{n/3}$. The probability that at least one of the $n/3$ 0-bits has never flipped during t steps is $1-(1-(1-1/n)^t)^{n/3}$. Hence, the probability that at least one 0-bit has not been flipped during $t=(n-1)\ln n$ steps is $1-(1-(1-1/n)^{(n-1)\ln n})^{n/3} \geq 1-e^{-1/3}$.

Altogether, the optimization time of (1+1) EA_b is $\Omega(n \log n)$ with probability at least $1-e^{-1/3}-e^{-\Omega(n)} = \Omega(1)$ which proves the theorem. □

5.3 Expected multiplicative weight decrease

The method of the expected multiplicative weight decrease has been developed to analyze the runtime behavior of randomized search heuristics until they compute a minimum spanning tree of a given graph. In the case of the minimum spanning tree problem, one considers the time until the weight of an arbitrary spanning tree has been decreased such that a minimum spanning tree has been achieved. The application of this method to the problem of computing minimum spanning trees can be found in Chapter 7. In this section we want to discuss the method and describe how to apply it. We show how to obtain in a simple way a polynomial upper bound on the optimization of linear pseudo-boolean function which is optimal in the case that the weights are polynomially bounded in n.

The method of the expected multiplicative weight decrease can be applied to problems where we are able to transfer each solution s into an optimal solution s_{opt} by a set $O=\{o_1,\ldots,o_r\}$ consisting of r operations that all have the same probability to happen in the next step. We assume that this probability can be lower bounded by α and that the set of possible fitness values contains only integers. For simplicity the value of r does not change in all considerations. Note that the number of operations until s_{opt} has been reached, depends on the solution s. W. l. o. g. we assume that $O'=\{o_1,\ldots,o_{r_1}\}$, $O' \subseteq O$, is the set of operations that is necessary to turn s into s_{opt}. Then $r-r_1$ operations are added such that one can work at each time step with the same value of r. It is important that the application of each of the operations of O' leads to a solution s' that is not inferior to s. This implies that each operation of O' applied to s is accepted. W. l. o. g. we assume that the considered fitness function f should be maximized. Let $d=f(s_{opt})-f(s)$ be the distance (measured in the difference of the function values) of s to an optimal one. As all operations have the same probability the expected decrease of the distance when producing a solution s' by an operation that is chosen uniformly at random from the set O is at least $\frac{f(s_{opt})-f(s)}{r}$. Note that non-accepted operations of $O \setminus O'$ contribute a distance decrease of 0. The expected distance of s' to s_{opt} is $(1-1/r)\cdot(f(s_{opt})-f(s))$ after

1 step, and the expected distance after t such steps is $(1-1/r)^t \cdot (f(s_{opt}) - f(s))$. Let $d_{max} = \max_{s \in \{0,1\}^n}(f(s_{opt}) - f(s))$ be the maximum distance of any search point in the search space to an optimal one. After having executed t randomly chosen operations of O the expected distance to an optimal solution is at most $(1-1/r)^t \cdot d_{max}$. Choosing $t = c \cdot r \cdot \log d_{max}$, c an appropriate constant, the expected distance is at most $1/2$. Using Markov's inequality (see Appendix A.2.1) the probability that the distance is at least 1 is upper bound by $1/2$. As the set of possible fitness values contains only integers the probability of having achieved an optimal solution (i. e. the distance is 0) is at least $1/2$. This implies that the expected number of operations belonging to the set O until an optimal solution has been achieved is at most $2t = O(r \cdot \log d_{max})$. The probability of an operation belonging to the set O is at least $r \cdot \alpha$. Using this the expected optimization time is $O((r \cdot \alpha)^{-1} r \cdot \log d_{max}) = O(\alpha \cdot \log d_{max})$.

We consider linear pseudo-boolean functions and define $w_{max} = \max_i |w_i|$. Applying the method of the expected multiplicative weight decrease, we show in a simple way an upper bound on the expected optimization time of (1+1) EA_b on each linear pseudo-boolean function which is due to Theorem 5.2.1 optimal as long as the weights are polynomially bounded in n. W. l. o. g. we assume that $w_i \geq 0$, $1 \leq i \leq n$, holds.

Theorem 5.3.1 *The expected optimization time of (1+1) EA_b on linear functions is upper bounded by $O(n(\log n + \log w_{max}))$.*

Proof: The set of operations O contains all steps where only one single bit flips. Hence, O contains $r = n$ operations. The set O' contains all operations flipping one single 0-bit. As $w_i \geq 0$, $1 \leq i \leq n$, each operation of O' is accepted. The probability for one specific operation of O is $1/n \cdot (1-1/n)^{n-1} \geq 1/(en) := \alpha$ and $d_{max} \leq 2 \cdot n \cdot w_{max}$ holds. Using the method of the expected multiplicative weight decrease the expected optimization time is upper bounded by $O(n \log d_{max}) = O(n(\log n + \log w_{max}))$. □

Note that the given upper bound is $O(n \log n)$ as long as all weights are polynomially bounded in n. It is possible to obtain a more general upper bound of $O(n \log n)$ even if the weights are not polynomially bounded. This proof is much more complicated than the one presented here and can be found in Droste, Jansen, and Wegener (2002).

Part III

Single-Objective Optimization Problems

Chapter 6

Eulerian Cycles

The aim of this chapter is to start the analysis of randomized search heuristics on arc routing problems with respect to the expected time until they consider an optimal search point. We consider the well-known problem of computing an Eulerian cycle of a given undirected connected graph with n vertices and m edges (see Skiena (1990) for example). This is the simplest problem belonging to the wide class of arc routing problems. It can be solved in worst case run time of $O(m + n)$ by the algorithm of Hierholzer (1873).

As already argued, we do not and cannot hope to compete with the best algorithms for the Eulerian cycle problem. This can be different for generalizations of the problem. For example the problem of finding the largest Eulerian subgraph of a given graph and the mixed chinese postman problem (see Edmonds and Johnson (1973)) are NP-hard and evolutionary algorithms have a good chance to be competitive on these problems. For other NP-hard variants like the capacitated arc routing problem evolutionary algorithms have been developed and successfully applied (see e. g. Lacomme, Prins, and Ramdane-Chérif (2001)). Before we are able to analyze evolutionary algorithms on such problems, we think that it is important to understand evolutionary algorithms that work on simple arc routing problems.

In this chapter we consider two simple randomized search heuristics that use a permutation of the edges of the given graph to find an Eulerian cycle. The representation of permutations has successfully been applied to difficult combinatorial optimization problems like the traveling salesperson problem (see Michalewicz and Fogel (2004) for an overview). It is important to understand how evolutionary algorithms, using this encoding, work on simple problems. This is one step in the direction to analyze more complicated problems and algorithms using such an encoding. Starting with the first edge in the permutation, we can build up a path. One goal is to estimate the expected time until a path found by the algorithm is lengthened. If the current path is a cycle there may be $\Theta(m)$ operations necessary to lengthen this path. In such a case the algorithm has to walk on a plateau of constant fitness. Jansen and Wegener (2001) have studied how evolutionary algorithms can cope with plateaus in the binary search space. In the search space of permutations

plateaus have a different structure. Our analysis of the considered algorithms, using jumps in the mutation step, points out the structure of plateaus for the Eulerian cycle problem in the search space of permutations and how evolutionary algorithms can leave such a plateau. Changing the operator used for mutations, we point out that such a plateau can change to a local optimum with a large inferior neighborhood.

After having motivated to analyze randomized search heuristics on the Eulerian cycle problem, we describe in Section 6.1 our model of the problem. In Section 6.2, we show that RLS_p is able to compute an Eulerian cycle in expected polynomial time. The analysis is extended to (1+1) EA_p in Section 6.3. In Section 6.4 we consider algorithms using exchanges instead of jumps in the mutation step and prove that the expected optimization time in this case can be infinite or exponential. We finish with concluding remarks.

6.1 The problem

Euler initiated the study of graph theory in 1736 with the famous seven bridges problem. The generalization of the seven bridges problem can be described as follows and is known as the Eulerian cycle problem. Given an undirected connected graph $G = (V, E)$ on n vertices and m edges, the task is to compute a cycle such that every edge is used exactly one time. Euler proved that a tour of all edges in a connected undirected graph without repetition is possible iff the degree of each vertex is even. Such graphs are known as Eulerian graphs. If an Eulerian cycle exists we call G Eulerian. In the rest of this chapter we assume that G is Eulerian. To find such a cycle we use a permutation of the edges of G. The fitness of a permutation π is the length of the path implied by π when we start with the first edge of it.

The search space S_m contains all permutations of the edges of G. A search point $\pi \in S_m$ corresponds to the order to use the edges for the Eulerian tour. Usually a permutation does not correspond to an Eulerian tour. It normally describes a path p which is part of such a tour. The ideas can be used to define the fitness function *path* which is appropriate for the Eulerian cycle problem.

We will investigate the fitness function

$$path(\pi) := \textit{ length of the path implied by } \pi,$$

where we start with the first edge in π and extend the path, if the edge on the second position has one vertex with the first edge of π in common. This path can be further extended if the third edge has one vertex which is equal to the "free" vertex of the second edge. We can extend the procedure to build up a path of length l implied by π. In the rest of this chapter the path will be named by p and the part of π consisting of the positions $l + 1, \ldots, m$ will be named by q. Usually a path p is written as a sequence of vertices and denoted by $p = v_0, v_1, \ldots, v_l$. This implies a set of edges that are a subset of the edge set E. To make the connection to the fitness function *path* more precise we write

1. Find a cycle C in G
2. Delete the edges of C from G
3. If G is not empty go to step 1.
4. Construct the Eulerian cycle from the cycles produced in Step 1.

Figure 6.1: The algorithm of Hierholzer for the computation of Eulerian cycles

a path p by a sequence of directed edges and denote it by $p = (v_0, v_1), (v_1, v_2), \ldots, (v_{l-1}, v_l)$.

To our opinion the proposed fitness function is the most intuitional one that can be described for the Eulerian cycle problem. The fitness function describes the processing order to use the edges for a tour starting with the edge on position 1. Another advantage of this fitness function is that it can be easily evaluated. If the resulting path is short most of the edges in the permutation do not have to be considered.

For the Eulerian cycle problem algorithms have been designed that compute an Eulerian cycle in linear time. To analyze randomized search heuristics we use the knowledge that has been put into these algorithms. The algorithm (see Figure 6.1) proposed by Hierholzer (1873) computes an Eulerian cycle of a given Eulerian graph G and contains ideas which will be later used in the analysis of our algorithms.

Randomized search heuristics do not have the knowledge that the problem can be solved by computing cycles and building up the solution by putting the cycles together. We will see that they are able to compute a single cycle and integrate another cycle if the solution is not optimal. Hence, they follow the idea of the algorithm without having this global knowledge.

6.2 Analysis of RLS_p

In the following we show an upper bound of $O(m^5)$ on the expected optimization time for RLS_p on the proposed fitness function.

Theorem 6.2.1 *The expected time until RLS_p working on the fitness function path constructs an Eulerian cycle is $O(m^5)$.*

Proof: The fitness $path(\pi)$ of a search point π can take values from $\{1, \ldots, m\}$, where the optimum is reached if $path(\pi)$ equals m. W.l.o.g. the path p implied by π is of the form $(v_0, v_1), \ldots, (v_{l-1}, v_l)$ and has length $l \in \{1, \ldots m-1\}$. If $v_0 \neq v_l$ holds there is an

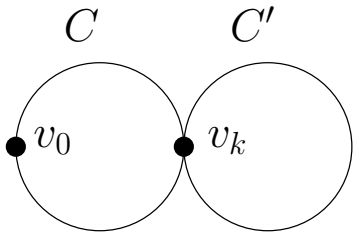

Figure 6.2: Situation in which p is a cycle C which does not include all edges of G. Then there is another cycle C' which has one vertex v_k with C in common.

edge incident to v_l which can be placed after $\{v_{l-1}, v_l\}$. Such a jump lengthens the path and has probability $\Omega(1/m^2)$. In this case the expected time for an improvement of π is bounded by $O(m^2)$.

If v_0 and v_l are equal the analysis for an improvement is more complicated. In this case p is a cycle C. If the graph is Eulerian and p is not an Eulerian tour there is at least one vertex v_k on C which is also a vertex on another cycle C' having v_k with C in common (see Figure 6.2). We want to show that a path p with length l can be lengthened to a path of length at least $l+1$ in expected time $O(m^4)$.

Claim 6.2.2 *Let p be a cycle described by π of length $l \in \{1, \ldots, m-1\}$. The expected time to produce a path of length at least $l+1$ is $O(m^4)$.*

Proof: We consider cm^4 steps of RLS_p, where c is a constant large enough. If we can lower bound the probability to produce a path of length at least $l+1$ in cm^4 steps by a positive constant α, the expected time for an improvement is at most $\frac{cm^4}{\alpha} = O(m^4)$.

We inspect the case where p is a cycle which corresponding permutation π does not start with the vertex v_k. Jumps that do not affect the positions $1, \ldots, l+1$ in π do not change the fitness value. Jumps which affect the positions 1 or $l+1$ have the possibility to lengthen the path and are essential for an improvement. If we have jumps that place elements of p, not at position 1 or l, at another position we get a shorter path. These steps are not accepted by the algorithm. The jump of the edge at position 1 to a position j, $j \neq l$, shortens the path by at least one and is also not accepted. The same holds for the jump of an edge at position $i \in \{l+1, \ldots, m\}$ to a position $j \in \{1, \ldots, l\}$. Hence, the algorithm only accepts the two jump operations $jump(1, l)$ and $jump(l, 1)$ if we do not have the possibility to reach an improvement by jumping to position 1 or $l+1$.

We call a mutation step relevant, if the current path p is changed and accepted. As we have exactly 2 jumps which are relevant we get a probability of $\frac{2}{m(m-1)}$ for a relevant step. The expected number of relevant steps in cm^4 steps is therefore $\frac{2cm^4}{m(m-1)}$. Thus, we can bound the probability to have less than $c'm^2$, c' a positive constant depending on c, relevant steps in cm^4 steps by $e^{-\Omega(m^2)}$.

We estimate the probability to construct a permutation in $M = c'm^2$ relevant steps, which starts with the edge $\{v_k, v_{k+1}\}$. To reach an improvement our aim to construct a path $p^* = (v_k, v_{k+1}), \ldots, (v_{l-1}, v_0), \ldots, (v_{k-1}, v_k)$. If we have not reached such a path there

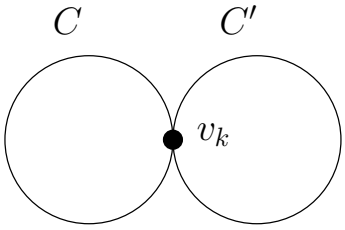

Figure 6.3: Instance G_m: Two cycles of length $\Theta(m)$ sharing one vertex

is exactly one jump which places $\{v_k, v_{k+1}\}$ one position further to the left and one which places $\{v_k, v_{k+1}\}$ one position further to the right. The probability to place $\{v_k, v_{k+1}\}$ further to the left is in each relevant step $\frac{1}{2}$. Let X be the number of steps to the left in M steps, and let b be an appropriate constant. The probability of exactly r steps to the left in M relevant steps equals

$$\begin{aligned} Pr(X = r) &= \binom{M}{r}\left(\frac{1}{2}\right)^r\left(1-\frac{1}{2}\right)^{M-r} \\ &\leq \binom{M}{M/2}2^{-M} \\ &\leq \frac{\sqrt{3\pi}e^{-M}M^{M+\frac{1}{2}}2^{-M}}{(\sqrt{2\pi}e^{-\frac{M}{2}}(M/2)^{M/2+1/2})^2} \qquad (Stirling) \\ &= b \cdot M^{-1/2} \\ &\leq \frac{1}{2m} \end{aligned}$$

The probability to have less than $\frac{M}{2}+\frac{m}{2}$ steps to the left is bounded above by $\frac{1}{2}+\frac{m}{2}\cdot\frac{1}{2m} = \frac{3}{4}$. Hence, we get a probability of at least $\frac{1}{4}$ to produce p^*.

As v_k is also a vertex in another cycle C' there are two edges $\{v_k, v_s\}$ and $\{v_k, v_t\}$ in C'. If we place one of these edges at position $l+1$ we have lengthened the path and achieved an improvement. On the other hand there are exactly two relevant jumps which place $\{v_k, v_{k+1}\}$ at a position other than position 1. This yields a probability of at least $\frac{1}{2}$ for an improvement in the next relevant step. Altogether we get a probability of at least $\alpha = 1 - \left(\frac{3}{4} + \frac{1}{4}\cdot\frac{1}{2} + o(1)\right) = \frac{1}{8} - o(1)$ to lengthen the path in cm^4 steps which implies an expected time for an improvement of $O(m^4)$. □

As there are at most $m-1$ improvements we get an upper bound of $O(m^5)$ for the expected optimization time of RLS_p. □

The most costly improvements are improvements where a cycle has to be revolved. There are instances of the problem where such an operation is essential for an improvement. We consider the graph $G_m = (V', E')$ (see Figure 6.3) consisting of n vertices and $m = n+1$ edges. The edge set consists of two cycles C und C', both of length $\Theta(m)$, having only the vertex v_k in common. G_m has the property that it leads for jump operations to large

plateaus of constant fitness and for exchange operations to local optima with large inferior neighborhoods.

Theorem 6.2.3 *Running RLS_p on the instance G_m an improvement revolving a cycle by $\Theta(m)$ operations is necessary with probability at least $\frac{1}{3} - o(1)$.*

Proof: After random initialization each edge of G_m has probability $1/m$ to be on the first position of π. As there are $o(m)$ edges which have a distance of $o(m)$ to v_k, the probability to have an edge with distance $o(m)$ to v_k is bounded above by $o(1)$. Hence, with probability $1 - o(1)$ we have at position 1 an edge which has a distance $\Omega(m)$ to v_k. W. l. o. g. we assume that the edge on the first position is part of the cycle $C = \{e_1, \ldots, e_r\}$, where $e_i \cap e_{i+1} \neq \emptyset$, $1 \leq i \leq r-1$, and $e_1 \cap e_r \neq \emptyset$ holds.

Each edge of G_m has at most 4 edges with which it has a vertex in common. The probability that the edges at position 1 und 2 have after initialization one vertex in common is bounded by $O(1/m)$. The probability to have a path of length at least 2 is therefore bounded by $O(1/m)$. Hence, we start with probability $1 - o(1)$ with a path of length 1 that has a distance $\Omega(m)$ to v_k. If we start with a path of length 1 we accept in the next step each jump operation. The expected time to produce a path p of length 2 is bounded by $O(m^2)$. As there are $o(m)$ edges which have a distance $o(m)$ to v_k, the probability to have a path p of length 2 which consists of such an edge is bounded by $o(1)$.

We consider the case where p has length 2. A path of length at least 3 is produced within $O(m^2)$ steps of the algorithm. Let e_i and e_{i+1} be the edges on positions 1 and 2 if we start with a path of length 2. The jump operations $jump(1,2)$ and $jump(2,1)$ are relevant but only improving if the edge on the third position of π is either e_{i-1} for $jump(1,2)$ or e_{i+2} for $jump(2,1)$. Such an operation can shorten the distance of the start or end vertex of p to v_k by at most 1. Furthermore, the jump of edge e_{i-1} to position 2 is relevant but only improving if the edge e_{i-2} is already on the third position. Each of these non improving jumps can shorten the distance of the start or end of p to v_k. The expected number of these relevant steps in $O(m^2)$ steps of the algorithm equals $\Theta(1)$. By Markov's inequality the probability to have $\omega(1)$ steps that shorten the distance to v_k is bounded by o(1).

If we have a path of length l, $l \geq 3$, that does not start or end with an edge containing v_k there are exactly two jumps (one jump to position 1 and one to position $l+1$) that lengthen the path. If p has length l and ends with the vertex v_k, there are three possibilities to lengthen the path at v_k. One jump that lengthens p by the other edge of C incident to v_k and two possibilities to lengthen the path by an edge of C'. Hence, the probability to use the edge of C for the extension of p at v_k is $\frac{1}{3}$. After this extension has at least 2 edges, we only accept changes of p if we lengthen the path. The cycle C having length r is produced within $O(r)$ improvements. As an improvement can only be reached by a jump to the first position or a jump to position $l+1$, the expected number of improvements changing the edge on the first postion of π equals $\frac{r}{2}$. By Chernoff bounds the probability to have more than $\frac{3r}{4}$ improvements with jump to position 1 is bounded by $e^{-\Omega(m)}$. Hence, the probability to construct a cycle C where start and end points have distance $\Omega(m)$ to the vertex v_k is at least $\frac{1}{3} - o(1)$. To improve the solution we have to revolve the cycle

by $\Theta(m)$ operations into one direction. Hence, an improvement revolving a cycle by $\Theta(m)$ operations is necessary with a probability of at least $\frac{1}{3} - o(1)$. □

6.3 Analysis of (1+1) EA_p

To analyze the expected optimization time of (1+1) EA_p on the fitness function *path* we have to consider the effect of more than one jump. We work under the assumption that p is a cycle which is not optimal. Otherwise, the probability of an improvement in the next step is at least $\frac{1}{em(m-1)}$, because one single jump can lengthen the path. This gives an expected time for an improvement of $O(m^2)$. To get an upper bound on the expected optimization time, it is important to analyze relevant mutations with $k > 1$ jumps that affect the part p of π. Again we assume that π does not start with a vertex v_k at which the path can be lengthened and bound the effect of accepted mutation steps that do not lead to an improvement. We say that we have a relevant mutation with k jumps, if the mutation changes p by k jumps and is accepted. First, we consider relevant mutations with more than 3 jumps in one mutation step. Later, we analyze relevant mutations with 2 and 3 jumps in one step.

Lemma 6.3.1 *Let p be a cycle, which is not an Eulerian cycle. The probability to have in $O(m^4)$ steps a relevant mutation, which changes p by at least 4 jumps and does not improve the fitness value, is $o(1)$.*

Proof: The probability to have a mutation with exactly k jumps equals $\frac{1}{e(k-1)!}$. Hence, the probability to have $k = \Omega(m)$ jumps in one mutation step is exponentially small and this also holds for a polynomial number of mutation steps. In the rest of this proof, we work under the assumption that $k = o(m)$ holds.

We consider the different cases of relevant jumps $jump(i, j)$ that change p. There may be other jumps in the part q of π, but these additional jumps only lead to a smaller probability.

If $i \in \{1, \ldots, l\}$ and $j \in \{1, \ldots l\}$ hold the jump has only influence on the path p. If we have 1 jump in the mutation step we get a probability $\Theta(1/m^2)$ for a relevant step that shifts p to the left or to the right. If there is more than one jump in one mutation step p is eventually shifted more than one position to the left or to the right. W. l. o. g. we assume that p is shifted to the left by putting the first r edges $e_1, \ldots e_r$ of p to the end of p. The first edge in this sequence of jumps has to jump to position l. Edge e_i, $i \geq 2$, has to jump to the position of e_j, where e_j is the edge with the greatest index smaller than i, that has already been placed at the end of p. If such an e_j has not yet been placed at the end of p, e_i has to jump to the position before the edge e_k, where k is the smallest index greater than i. If both edges e_j and e_k already exist at the end of p, the position to which e_i has to jump is for both cases the same. Hence, for each edge e_i there is exactly one possible position to jump to and we can estimate the probability of a success in one step by

$$\frac{r!}{e(r-1)!(m(m-1))^r} = \frac{r}{e(m(m-1))^r}.$$

If $r \geq 3$ the probability of such a mutation in $O(m^4)$ steps is bounded by $O(1/m^2)$.

If p contains cycles of the kind $c = \{\{u_1, u_2\}, \{u_2, u_3\}, \ldots, \{u_t, u_1\}\}$ there is another possibility to have sequences of relevant jumps that are not improving. Such a mutation is accepted if it changes the order of cycles in p. As we are considering undirected graphs cycles have length at least 3. A sequence of $t \geq 3$ jumps is accepted if it displaces a cycle c from p at another position of p. There are $O(l)$ possible positions where c can be placed. The number of possible cycles in p is bounded by $O(l)$. We inspect the displacement of a cycle c with length t in greater detail. The cycle c has to "jump over" another cycle, so we have to execute at least t jumps. There are $t!$ possibilities to order the t edges in c with respect to the jumps. W. l. o. g. we assume that c is displaced to the right in p. Let e_i be the edge in the first jump. As we want to have a cycle after the mutation step, e_i has to be adjacent to u_1 after the first jump. This is only possible if we jump e_i to a position that contains an edge with a vertex u_1. The second jump of our sequence determines how to traverse the cycle. The edge can be placed before or behind e_i. As the direction to traverse the cycle is determined by the second jump, the positions for the other edges to jump to are fixed. Hence, the probability to displace a cycle at another fixed position in p is at most

$$\frac{2 \cdot t!}{e(t-1)!(m(m-1))^t} = \frac{2 \cdot t}{e(m(m-1))^t}.$$

As the number of cycles as well as the number of possible positions is bounded by $O(l)$ the probability to displace a cycle of length $t = \Theta(1)$ in $O(m^4)$ steps is bounded by $O\left(\frac{1}{m^{2(t-3)}}\right)$. If $t = \omega(1)$ holds, this probability is bounded by $O\left(\frac{1}{m^{2(t-3)-1}}\right)$. It follows that the probability to have a mutation in $O(m^4)$ steps, which displaces a cycle of length $t \geq 4$, is bounded above by $O(1/m^2)$.

There may be combinations of shifting p to the left or to the right and a displacement of a cycle. Such a mutation has to consist of a displacement of at least one cycle of length 3 and at least one jump operation that shifts p to the left or to the right. For such an event we get a probability of $O(1/m^2)$ in $O(m^4)$ steps because there are only two possibilities for the shifting operation.

We examine the case where $i \in \{1, \ldots, l\}$ and $j \in \{l+1, \ldots m\}$ hold. In this case an edge from p jumps to a position of q. p describes a cycle iff it displaces a cycle from p. Consider a cycle c with t edges that should be displaced from p. For each edge there are $O(m-l)$ position to jump to. Hence, the probability to displace the edges of c is upper bounded by

$$\frac{t!(m-l)^t}{e(t-1)!(m(m-1))^t} = \frac{t(m-l)^t}{e(m(m-1))^t)}.$$

For each $t \geq 3$ this is bounded by $O(1/m^3)$. As there are at most $O(l)$ cycles that can be displaced from p, the probability to displace a cycle from p in the next step is bounded by $O(1/m^2)$. To accept such a displacement, edges of q have to be integrated into p and form a cycle. There are $O(m-l)$ cycles that can be generated from the edges of q. Cycles can be generated at $O(l)$ positions of p. Hence, the probability that this happens in the next step is bounded by

$$\frac{O(l)O((m-l))t!}{e(t-1)!(m(m-1))^t} = \frac{t}{\Omega(m^{2(t-1)})}.$$

For each $t \geq 3$ this is upper bounded by $O(1/m^4)$. Thus, the probability to displace one cycle from p and integrate a new cycle into p in $O(m^4)$ steps is bounded by $\frac{O(m^4)}{\Omega(m^4)\Omega(m^2)} = O(1/m^2)$.

If $i \in \{l+1,\ldots,m\}$ and $j \in \{1,\ldots l\}$ hold there is an edge that jumps from q to p. Such a sequence of jumps can only be accepted if it includes a new cycle c' into p. To have a relevant mutation that does not lead to an improvement, the fitness value has to be unchanged. To achieve this we have to displace a cycle from p to q or to displace the edge which is after the integration of c' at position $l+1$. The displacement of the edge at position $l+1$ can only be done by the operations $jump(*, l+1)$ and $jump(l+1, *)$ where $* \in \{l+2,\ldots,m\}$ holds. The probability of such a jump equals $\frac{2(m-l-1)}{em(m-1)} = O(1/m)$. Hence, the probability to integrate a new cycle into p and displace one cycle of p in $O(m^4)$ steps is bounded by $O(1/m^2)$ and the probability to integrate one cycle and displace the edge at position $l+1$ is bounded by $O(1/m)$.

As the probability for each $k \geq 4$, $k = o(m)$, to have k relevant jumps in $O(m^4)$ mutation steps is bounded by $O(1/m)$, we can bound the probability to have a mutation with 4 or more jumps that is not improving in $O(m^4)$ steps by $o(m) \cdot O(1/m) = o(1)$. □

It remains to examine the effect of relevant mutations with 2 or 3 jumps that change p. Our aim is to bound the expected number of such mutations in $O(m^4)$ steps from above by a constant. Then we will see later that (1+1) EA_p is able to compensate the effect within $O(m^4)$ steps.

Lemma 6.3.2 *Let p be a cycle, which is not an Eulerian cycle. The expected number of non improving relevant mutations in $O(m^4)$ steps with 2 or 3 jumps that affect p is bounded by $O(1)$ with probability $1-o(1)$.*

Proof: We only have to consider the case where $i \in \{1,\ldots l\}$ and $j \in \{1,\ldots,l\}$ holds for $jump(i,j)$, because the other cases have already been considered in Lemma 6.3.1. If we have 2 jumps in one step we can only shift the beginning of p. It follows that the expected number of mutations is $\frac{O(m^4)}{\Omega(m^4)} = O(1)$. If we have 3 jumps in one step we can displace a cycle of length 3 or shift the beginning of p. Taking the arguments of the proof of the previous lemma into account, we get an expected number of mutations that displace a cycle of length 3 or shift the beginning of p by 3 jumps of $O(1)$. By Markov's inequality the probability to have $\omega(1)$ mutations of that kind is bounded by $1/w(1) = o(1)$. □

We have seen that the effect of more than one jump operation is small with probability $1-o(1)$. This can be used to prove an upper bound on the expected runtime of (1+1) EA_p.

Theorem 6.3.3 *The expected time until (1+1) EA_p working on the fitness function path constructs an Eulerian cycle is bounded by $O(m^5)$.*

Proof: We consider the path p implied by the current search point π. If it is not a cycle, the probability of an improvement in the next step is at least $\frac{1}{em(m-1)}$ which leads to an expected time for an improvement of $O(m^2)$. If p is a cycle C and p is not optimal there is another cycle C' that has one vertex v_k with C in common. The probability to have a relevant mutation with 4 or more jumps that is not improving in $O(m^4)$ steps is bounded by $o(1)$. Non improving relevant mutations with 2 or 3 jumps shift the edge $\{v_k, v_{k+1}\}$ in $O(m^4)$ steps O(1) positions away from position 1 with probability $1 - o(1)$.

Considering cm^4 steps we have at least $M = c'm^4$ relevant mutations consisting of one jump. If we have at least $\frac{M}{2} + \frac{m}{2} + O(1)$ mutations that shift $\{v_k, v_{k+1}\}$ to the left we reach a path p^* that starts with $\{v_k, v_{k+1}\}$. The probability that we do not reach p^* is bounded by $\frac{1}{2} + \frac{1}{4} + O(1/m)$. Hence, the probability to reach p^* is at least $\frac{1}{4} - o(1)$ and the probability that the next relevant step is improving is at least $\frac{1}{2} - o(1)$. It follows that the probability to have an improvement in cm^4 steps is at least $\frac{1}{8} - o(1)$ which leads to an expected time of $O(m^4)$ for an improvement. As the number of improvements until an Eulerian cycle has been constructed is bounded by $O(m)$, the expected optimization time until (1+1) EA_p finds an Eulerian cycle is $O(m^5)$. □

As in the case of RLS_p, improvements revolving a cycle $\Theta(m)$ steps into one direction dominate the runtime of (1+1) EA_p. We consider the instance G_m described in Section 6.2 to prove that such steps are necessary.

Theorem 6.3.4 *Running (1+1) EA_p on the instance G_m, an improvement revolving a cycle by $\Theta(m)$ operations is necessary with probability at least $\frac{1}{3} - o(1)$.*

Proof: As the expected time for an improvement is bounded by $O(m^2)$ if p is a path, we construct an Eulerian cycle or one of the cycles C and C' in $O(m^3)$ steps. The probability of a relevant step in $O(m^3)$ steps of the algorithm, which changes p by $t \geq 2$ jumps is bounded by $\frac{O(m^3)}{O(m^{2t})} = O(1/(m^{2t-3}) = O(1/m)$. As the probability to have a mutation consisting of $\Omega(m)$ jumps is exponentially small, the probability to have a relevant mutation with more than 1 jump in $O(m^3)$ steps is bounded by $o(m) \cdot O(1/m) + o(1) = o(1)$ from above. Hence, (1+1) EA_p behaves with probability $1 - o(1)$ like RLS_p which shows the theorem. □

6.4 Mutation using exchange operations

Many evolutionary algorithms based on the encoding of permutations use an exchange operation for mutation. The operation $exchange(i, j)$ executed on a permutation π consists of exchanging the elements at position i and j in π. The algorithms RLS_p^* and (1+1) EA_p^* use exchanges instead of jumps. Note, that an exchange operation $exchange(i, j)$, $i < j$, can be simulated by the two jumps $jump(i, j)$ and $jump(j - 1, i)$. The exchange of two elements i and j has no affect on the positions of other elements in the permutation.

We have shown in Sections 6.2 and 6.3 that RLS_p and (1+1) EA_p have to walk on a plateau of size $\Theta(m)$ with a probability bounded below by a constant. Considering RLS_p^*

and (1+1) EA^*_p this plateau changes to a local maximum with a large neighborhood having a smaller fitness value. In the following we will show that this leads to an infinite expected optimization time for RLS^*_p and an exponential optimization time for (1+1) EA^*_p on the instance G_m.

Theorem 6.4.1 *RLS^*_p working on the fitness function path has an infinite expected optimization time on G_m.*

Proof: W. l. o. g. we assume the path starts with an edge of C. With probability $1 - o(1)$ the path p has length 1 and the edge on the first position of π has distance $\Theta(m)$ to v_k. After the next improvement p has distance $\Theta(m)$ with probability $1 - o(1)$, because there are $o(m)$ edges having distance $o(m)$ to the vertex v_k.

If p is a path of length l that does not contain the vertex v_k, there is exactly one exchange operation exchanging the edge at position $l + 1$ with another edge of q that lengthens the path. If p ends with an edge containing v_k, there are exactly 3 possibilities to lengthen the path. One of them lengthens the path by an edge of C. Each of these three possibilities results in an extension of $\Theta(m)$ edges. Hence the probability to use the extension of C is at least $\frac{1}{3} - o(1)$. If we use this extension we construct the cycle C in an expected number of $O(m^3)$ steps. After that the start and end points of p have distance $\Omega(m)$ to v_k with probability at least $\frac{1}{3} - o(1)$. If this happens there is no relevant mutation, because each exchange operation affecting one edge of the cycle destroys the cycle and shortens the path. Hence, the optimization time is infinite with probability at least $\frac{1}{3} - o(1)$ which yields the theorem. □

(1+1) EA^*_p using exchanges instead of jumps has the possibility to leave such a trap. But this only happens with a small probability. We show that (1+1) EA^*_p has an exponential expected optimization time on the fitness function *path*.

Theorem 6.4.2 *(1+1) EA^*_p working on the fitness function path has an exponential expected optimization time on G_m.*

Proof: As RLS^*_p the algorithm (1+1) EA^*_p constructs p describing a cycle which is not Eulerian and whose start vertex has distance $\Omega(m)$ to v_k in $O(m^3)$ steps with probability at least $\frac{1}{3} - o(1)$.

W. l. o. g. we assume that the cycle C has been constructed. An $exchange(i, j)$ only affects the positions i and j. Edges at other positions are left unchanged. There are only two possibilities to construct another path p' that differs from p and is accepted. The first possibility is to change the position of at least one edge in p. To have a relevant step that changes the position of at least one edge, we have to displace all the other edges of C. Hence, for such a step $\Theta(m)$ exchange operations are necessary. The probability that this happens in one step is exponentially small.

The second possibility is to integrate the circle C' into the path. This is only possible if all the edges of C' are placed between the two edges containing the vertex v_k in p. As C' has length $\Theta(m)$ the probability that this happens in the next step is exponentially small. Hence, the probability that the next step is accepted is exponentially small, if we have reached the local optimum. The local optimum is reached with a probability at least $\frac{1}{3} - o(1)$ and this proves the theorem. □

6.5 Conclusions

The Eulerian cycle problem is a fundamental problem in graph theory belonging to the class of arc routing problems. Several important problems belonging to this class are difficult, and evolutionary algorithms have a good chance to be competitive on these problems. The analysis shows how evolutionary algorithms based on the encoding of permutations work. This is also important because many evolutionary algorithms based on this encoding have been proposed for the traveling salesman problem and different NP-hard scheduling problems.

As a first step towards the analysis of evolutionary algorithms for the mentioned problems, upper bounds on the expected runtime of RLS_p and (1+1) EA_p for the Eulerian cycle problem have been proven. We have also presented an instance where the most costly improvements revolving a cycle by $\Theta(m)$ operations are necessary for both algorithms with probability at least $\frac{1}{3} - o(1)$. Such investigations have led to the result that the variants RLS_p^* and (1+1) EA_p^*, using exchanges for mutation, have an infinite or exponential optimization time on this instance.

A conference version that contains the results of this chapter has been published in the Proceedings of the Congress on Evolutionary Computation (CEC) 2004 (see Neumann (2004a)). A journal version has been submitted for publication.

Chapter 7

Minimum Spanning Trees

In this chapter we study the behavior of randomized search heuristics on another important combinatorial optimization problem. We consider the well-known problem of computing a minimum spanning tree in a given undirected connected graph with n vertices and m edges. The problem has many applications in the area of network design. Assume that we have n computers that should be connected with minimum cost, where costs of a certain amount occur when one computer is connected to another one. The cost for a connection can for example be the distance between two considered computers. One needs to make $n-1$ connections between these computers such that all computers are able to communicate with each other. Considering a graph as a model for a possible computer network the graph has n vertices and one searches for the set of edges with minimal cost that makes the graph connected. The problem can be solved easily by greedy algorithms.

The famous algorithms due to Kruskal (1956) and Prim (1957) have worst-case run times of magnitude $O((n+m)\log n)$ and $O(n^2)$, respectively, see any textbook on efficient algorithms, e.g., Cormen, Leiserson, Rivest, and Stein (2001). Karger, Klein, and Tarjan (1995) have given a randomized greedy algorithm that computes a minimum spanning tree in time $O(m)$ with high probability. Greedy algorithms use global ideas. Considering only the neighborhoods of two vertices u and v, it is not possible to decide whether the edge $\{u,v\}$ belongs to some minimum spanning tree. Therefore, it is interesting to analyze the run times obtainable by more or less local search heuristics like randomized local search and evolutionary algorithms. One goal is to estimate the expected time until a better spanning tree has been found. For large weights, there may be exponentially many spanning trees with different weights, which means that the distance from a starting solution to an optimal one may be exponentially. Then it is important how much progress a randomized search heuristics can make with respect to an optimal solution. Therefore, we have to analyze how much better the better spanning tree is. This is indeed the first time the expected fitness increase is estimated for problems of combinatorial optimization. Based on this approach Witt (2005) has presented analyses for randomized search heuristics and the NP-hard partition problem.

As already argued, we do not and cannot hope to beat the best algorithms for the minimum spanning tree problem. This can be different for two generalizations of the problem. First, one is interested in minimizing the weight of restricted spanning trees, e.g., trees with bounded degree or trees with bounded diameter. These problems are NP-hard, and evolutionary algorithms are competitive, see Raidl and Julstrom (2003). Second, one is interested in the multi-objective variant of the problem. Each edge has k weights, and one looks for the Pareto optimal spanning trees with respect to the weight functions, see Hamacher and Ruhe (1994) for the general problem and Zhou and Gen (1999) for the design of evolutionary algorithms. Many polynomially solvable problems have NP-hard multi-objective counterparts, see Ehrgott (2000). None of these papers contains a run time analysis of the considered search heuristics. We think that it is essential to understand how the heuristics work on the unrestricted single-objective problem before one tries to analyze their behavior on the more difficult variants. Our results obtained in this chapter are the basis for the analysis of EAs on the multi-objective minimum spanning tree problem which we will present in Chapter 9.

After having motivated the problem to analyze randomized search heuristics on the minimum spanning tree problem, we give a survey on the rest of this chapter. In Section 7.1, we describe our model of the minimum spanning tree problem. The theory on minimum spanning trees is well established. In Section 7.2, we deduce some properties of local changes in non-optimal spanning trees which are applied in the run time analysis presented in Section 7.3. Some generalizations of the results obtained are presented in Section 7.4

7.1 The problem

This classical optimization problem has the following description. Given an undirected connected graph $G = (V, E)$ on n vertices and m weighted edges, find an edge set $E' \subseteq E$ of minimal weight, which connects all vertices. The weight of an edge set is the sum of the weights of the considered edges. Weights are positive integers. Therefore, the solution is a tree on V, a so-called spanning tree. One can also consider graphs which are not necessarily connected. Then the aim is to find a minimum spanning forest, i.e., a collection of spanning trees on the connected components. All our results hold also in this case. To simplify the presentation we assume that G is connected.

There are many possibilities how to choose the search space for randomized search heuristics. This problem has been investigated intensively by Raidl and Julstrom (2003). Their experiments point out that one should work with "edge sets". The search space equals $S = \{0,1\}^m$, where each position corresponds to one edge. A search point $s \in S$ corresponds to the choice of all edges e_i, $1 \leq i \leq m$, where $s_i = 1$. In many cases, many search points correspond to non-connected graphs and others correspond to connected graphs with cycles, i.e., graphs which are not trees. If all graphs which are not spanning trees get the same "bad" fitness, it will take exponential time to find a spanning tree when

we apply a general search heuristic. We will investigate two fitness functions w and w'. The weight of e_i is denoted by w_i. Let $w_{\max}$ be the maximum weight. Then $w_{\text{ub}} := n^2 \cdot w_{\max}$ is an upper bound on the weight of each edge set. Let

$$w(s) := (c(s) - 1) \cdot w_{\text{ub}}^2 + (e(s) - (n-1)) \cdot w_{\text{ub}} + \sum_{i|s_i=1} w_i$$

be the first fitness function where $c(s)$ is the number of connected components of the graph described by s and $e(s)$ is the number of edges in this graph. The fitness function has to be minimized and takes the weight of all edges into account for which corresponding bit $s_i = 1$ holds. The most important issue is to decrease $c(s)$ until we have graphs connecting all vertices. Then we have at least $n-1$ edges, and the next issue is to decrease $e(s)$ under the condition that s describes a connected graph. Hence, we look for spanning trees. Finally, we look for minimum spanning trees.

It is necessary to penalize non-connected graphs since the empty graph has the smallest weight. However, it is not necessary to penalize extra connections since breaking a cycle decreases the weight. Therefore, it is also interesting to investigate the fitness function

$$w'(s) := (c(s) - 1)w_{\text{ub}} + \sum_{i|s_i=1} w_i.$$

The fitness function w' is appropriate in the black-box scenario where the scenario contains as little problem-specific knowledge as possible. The fitness function w contains the knowledge that optimal solutions are *trees*. This simplifies the analysis of search heuristics. Therefore, we always start with results on the fitness function w and discuss afterwards how to obtain similar results for w'.

7.2 Properties of local changes of spanning trees

The theory on minimum spanning trees is well established. Here we want to show how an arbitrary spanning tree can be turned into an optimal solution in a specific way that can be used later for analyzing the runtime of randomized search heuristics. We identify a tree T by its set of edges. Let $e \in E \setminus T$ be an edge that is not contained in T. We denote by $Cyc(T, e)$ the edges of T that are contained in the cycle which is created when introducing e into T. It is well known that we can construct from a spanning tree T another spanning tree T' by introducing an edge $e \in E \setminus T$ into T and removing one edge of $Cyc(T, e)$ from T. Such operations are called exchange operations. In this section we recall some facts from the theory of minimum spanning trees that show that an arbitrary spanning tree T can be turned into an optimal solution T^* by a set of exchange operations where each operation is directly applicable on T and the execution of the operation does not lead to a weight increase. Using this we can estimate the weight decrease that is possible when considering the current spanning tree T.

The following result has been proven by Kano (1987) using an existence proof. Later Mayr and Plaxton (1992) have given an explicit construction procedure which we present in the following.

Theorem 7.2.1 *Let T be a minimum spanning tree and S be an arbitrary spanning tree of a given weighted graph $G = (V, E)$. Then there exists a bijection Φ from $T \setminus S$ to $S \setminus T$ such that for every edge $e \in T \setminus S$, $\Phi(e) \in Cyc(S, e)$ and $w(\Phi(e)) \geq w(e)$.*

Proof: Let C and D be disjoint subsets of E. The graph $G' = G[C, D]$ is constructed from G by contracting the edges of C and deleting the edges of D. We determine the bijection between the disjoint spanning trees $T' = T \setminus S$ and $S' = S \setminus T$ of the graph $G' = G[T \cap S, E \setminus T \setminus S]$. It is easy to see that $Cyc(T', e) \subseteq Cyc(T, e)$ holds for all $e \in T'$. Let t be the heaviest edge in T' and s be any edge in S' for which $t \in Cyc(T', s)$ and $s \in Cyc(S', t)$ holds. We can determine such an s by removing t from G'. This partitions the vertices of T' into two classes. Let s be the edge in S' that connects these two components. Note that $s \in Cyc(S', t)$ and $t \in Cyc(T', s)$ holds as s and t connect the two components of $T' \setminus \{t\}$.

T' is a minimum spanning tree of G' which implies that $w(t) \leq w(s)$. Set $\Phi(t) = s$ and determine the next component of the bijection by repeating the procedure on the graph $G'[s, t]$. $T'[s, t]$ is a minimum spanning tree of $G'[s, t]$. In addition for all $e \in T'[s, t]$,

$$Cyc(S'[s, t], e) = Cyc(S', e) \setminus \{s\} \subseteq Cyc(S', e) \subseteq Cyc(S, e)$$

holds. Hence, the next assignment of an edge $e \in T'[s, t]$ to Φ will be guaranteed to satisfy $\Phi(e) \in Cyc(S, e)$. The process is iterated until for each $e \in T'$ a corresponding $\Phi(e) \in S'$ has been determined. □

Note, that Theorem 7.2.1 gives a set of edge exchanges to transform an arbitrary spanning tree into a minimum spanning tree.

We denote by w_{opt} the weight of minimum spanning trees and want to show the following. For a non-optimal search point s, there are either many weight-decreasing local changes which, on the average, decrease $w(s)$ by an amount which is not too small with respect to $w(s) - w_{\text{opt}}$, or there are few of these local changes which, on the average, cause a larger decrease of the weight. This statement will be made precise in the following lemma.

Lemma 7.2.2 *Let s be a search point describing a non-minimum spanning tree T. Then there exist some $k \in \{1, \ldots, n-1\}$ and k different accepted 2-bit flips such that the average weight decrease of these flips is at least $(w(s) - w_{\text{opt}})/k$.*

Proof: Let s^* be a search point describing a minimum spanning tree T^*. Let $k := |T^* \setminus T|$. Then there exists a bijection $\Phi : T^* \setminus T \to T \setminus T^*$ such that $\Phi(e)$ lies on the cycle $Cyc(T, e)$ and the weight of $\Phi(e)$ is not smaller than the weight of e due to Theorem 7.2.1.

We consider the k 2-bit flips flipping e and $\Phi(e)$ for $e \in T^* \setminus T$. They are accepted since e creates a cycle which is destroyed by the elimination of $\Phi(e)$. Performing all the k

2-bit flips simultaneously changes T into T^* and leads to a weight decrease of $w(s) - w_{\text{opt}}$. Hence, the average weight decrease of these steps is $(w(s) - w_{\text{opt}})/k$. □

The analysis performed in Section 7.3 will be simplified if we can ensure that we always have the same parameter k in Lemma 7.2.2. This is easy if we allow also non-accepted 2-bit flips whose weight decrease is defined as 0. We add $n - k$ non-accepted 2-bit flips to the set of the k accepted 2-bit flips whose existence is proven in Lemma 7.2.2. Then we obtain a set of exactly n 2-bit flips. The total weight decrease is at least $w(s) - w_{\text{opt}}$ since this holds for the k accepted 2-bit flips. Therefore, the average weight decrease is bounded below by $(w(s) - w_{\text{opt}})/n$. We state this result as Lemma 7.2.3.

Lemma 7.2.3 *Let s be a search point describing a spanning tree T. Then there exists a set of n 2-bit flips such that the average weight decrease of these flips is at least $(w(s)-w_{\text{opt}})/n$.*

When analyzing the fitness function w' instead of w, we may accept non-spanning trees as improvements of spanning trees. Non-spanning trees can be improved by 1-bit flips eliminating edges of cycles. A 1-bit flip leading to a non-connected graph is not accepted and its weight decrease is defined as 0.

Lemma 7.2.4 *Let s be a search point describing a connected graph. Then there exist a set of n 2-bit flips and a set of $m-(n-1)$ 1-bit flips such that the average weight decrease of these flips is at least $(w(s) - w_{\text{opt}})/(m+1)$.*

Proof: We consider all 1-bit flips concerning the edges that are not contained in the minimum spanning tree T^*. If we try them in some arbitrary order we obtain a spanning tree T. If we consider their weight decrease with respect to the graph G' described by s, this weight decrease can be only larger. The reason is that a 1-bit flip, which is accepted in the considered sequence of 1-bit flips, is also accepted when applied to s. Then we apply Lemma 7.2.3 to T. At least the same weight decrease is possible by adding e_i and deleting a non-T^* edge with respect to G'. Altogether, we obtain at least a weight decrease of $w(s) - w_{\text{opt}}$. This proves the lemma, since we have chosen $m+1$ flips. □

7.3 The analysis of RLS_b and (1+1) EA_b

The fitness functions w penatalizes solutions that are not connected or have more than $n-1$ edges. In the case of w' unconnected graphs get high fitness values as the number of connected components is multiplied by a large amount. We first show that RLS_b and (1+1) EA_b using the fitness functions w and w' construct connected graphs efficiently. In the case of w it is also easy to show that spanning trees are obtained in a small amount of time. The technique used for these analyses is based on partitioning the search space into fitness levels with respect to the number of connected components respectively to the number of edges for a fixed number of connected components.

Lemma 7.3.1 *The expected time until RLS$_b$ or (1+1) EA$_b$ working on one of the fitness function w or w' has constructed a connected graph is $O(m \log n)$.*

Proof: The fitness functions are defined in such a way that the number of connected components will never be increased in accepted steps. For each edge set leading to a graph with k connected components, there are at least $k-1$ edges whose inclusion decreases the number of connected components by 1. Otherwise, the graph would not be connected. The probability of a step decreasing the number of connected components is at least $\frac{1}{2} \cdot \frac{k-1}{m}$ for RLS$_b$ and $\frac{1}{e} \cdot \frac{k-1}{m}$ for (1+1) EA$_b$. Hence, the expected time until s describes a connected graph is bounded above by

$$em\left(1 + \cdots + \frac{1}{n-1}\right) = O(m \log n). \qquad \square$$

Lemma 7.3.2 *If s describes a connected graph, the expected time until RLS$_b$ or (1+1) EA$_b$ has constructed a spanning tree for the fitness function w is bounded by $O(m \log n)$.*

Proof: The fitness function w is defined in such a way that, starting with s, only connected graphs are accepted and that the number of edges does not increase. If s describes a graph with N edges, it contains a spanning tree with $n-1$ edges, and there are at least $N-(n-1)$ edges whose exclusion decreases the number of edges. If $N = n-1$, s describes a spanning tree. Otherwise, by the same arguments as in the proof of Lemma 7.3.1, we obtain an upper bound of

$$em\left(1 + \cdots + \frac{1}{m-(n-1)}\right) = O(m \log(m-n+1)) = O(m \log n). \qquad \square$$

This lemma holds also for RLS$_b$ and the fitness function w'. RLS$_b$ does not accept steps only including an edge or only including two edges if s describes a connected graph. Since RLS$_b$ does not affect more than two edges in a step, it does not accept steps in which the number of edges of a connected graph is increased. This does not hold for (1+1) EA$_b$. It is possible that the exclusion of one edge and the inclusion of two or more edges creates a connected graph whose weight is not larger than the weight of the given graph.

Before we analyze the expected time to turn a spanning tree into a minimum spanning tree, we investigate an example (see Figure 7.1). The example graph consists of a connected sequence of p triangles and the last triangle is connected to a complete graph on q vertices. The number of vertices equals $n := 2p+q$ and the number of edges equals $m := 3p+q(q-1)/2$. We consider the case of $p = n/4$ and $q = n/2$ implying that $m = \Theta(n^2)$. The edges in the complete graph have the weight 1 and we set $a := n^2$. Each triangle edge has a weight which is larger than the weight of all edges of the complete graph altogether. Theorem 7.3.3 and Theorem 7.3.8 prove that this graph is a worst-case instance with polynomial weights.

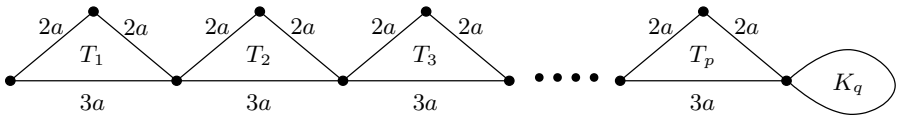

Figure 7.1: An example graph with p connected triangles and a complete graph on q vertices with edges of weight 1.

Theorem 7.3.3 *The expected optimization time until RLS_b and (1+1) EA_b find a minimum spanning tree for the example graph of Figure 7.1 equals $\Theta(m^2 \log n) = \Theta(n^4 \log n)$ with respect to the fitness functions w and w'.*

Proof: The upper bound is contained in Theorem 7.3.8. Here we prove the lower bound by investigating typical runs of the algorithm. We partition the graph G into its triangle part T and its clique part C. Each search point x describes an edge set. We use the following notation.

- $d(x)$: number of disconnected triangles with respect to the edges chosen by x
- $b(x)$: number of bad triangles (exactly one $2a$-edge and the $3a$-edge are chosen)
- $g(x)$: number of good triangles (exactly the two $2a$-edges are chosen)
- $c(x)$: number of complete triangles (all three edges are chosen)
- $\mathrm{con}_G(x)$: number of connected components in the graph
- $\mathrm{con}_C(x)$: number of connected components in the clique part C of the graph
- $\mathrm{con}_T(x)$: number of connected components in the tree part T of the graph

We investigate four phases of the search. The first phase of length 1 is the initialization step producing the random edge set x. In the following, all statements hold with probability $1 - o(1)$.

Claim 7.3.4 *After initialization, $b(x) = \Theta(n)$ and $\mathrm{con}_C(x) = 1$.*

Proof: The statements can be proved independently since the corresponding parts of x are created independently. The probability that a given triangle is bad equals $1/4$. There are $n/4$ triangles and $b(x) = \Theta(n)$ by Chernoff bounds. We consider one vertex of C. It has $n/2 - 1$ possible neighbors. By Chernoff bounds, it is connected to at least $n/6$ of these vertices. For each other vertex, the probability to be not connected to at least one of these $n/6$ vertices is $(1/2)^{n/6}$. This is unlikely even for one of the remaining vertices. Hence, $\mathrm{con}_C(x) = 1$. □

For the following phases, we distinguish the steps by the number k of flipping triangle edges and call them k-steps. Let p_k be the probability of a k-step. For RLS_b, $p_1 = \Theta(n^{-1})$, $p_2 = \Theta(n^{-2})$ and $p_k = 0$, if $k \geq 3$. For (1+1) EA_b and constant k

$$p_k = \binom{3n/4}{k} \left(\frac{1}{m}\right)^k \left(1 - \frac{1}{m}\right)^{3n/4-k} = \Theta(n^k m^{-k}) = \Theta(n^{-k}).$$

For a phase of length $n^{5/2}$, the following statements hold. The number of 1-steps equals $\Theta(n^{3/2})$, the number of 2-steps equals $\Theta(n^{1/2})$, and there is no k-step, $k \geq 3$.

Claim 7.3.5 *Let $b(x) = \Theta(n)$ and $\mathrm{con}_C(x) = 1$. In a phase of length $n^{5/2}$, a search point y where $b(y) = \Theta(n)$ and $\mathrm{con}_G(y) = 1$ is produced.*

Proof: By Lemma 7.3.1, the probability of creating a connected graph is large enough. Let y be the first search point where $\mathrm{con}_G(y) = 1$. We prove that $b(y) = \Theta(n)$. All the 2-steps can decrease the b-value by at most $O(n^{1/2})$. A 1-step has two possibilities to destroy a bad triangle.

- It may destroy an edge of a bad triangle. This increases the con_G-value. In order to accept the step, it is necessary to decrease the con_C-value.
- It may add the missing edge to a bad triangle. This increases the weight by at least $2a$. No triangle edge is eliminated in this step. In order to accept the step, it is necessary to decrease the con_C-value.

However, $\mathrm{con}_C(x) = 1$. In order to decrease this value, it has to be increased before. A step increasing the con_C-value can be accepted only if the con_T-value is decreased in the same step at least by the same amount. This implies that triangle edges have to be added. For a 1-step, the total weight is increased without decreasing the con_G-value and the step is not accepted. Hence, only the $O(n^{1/2})$ 2-steps can increase the con_C-value. By Chernoff bounds, the number of clique edges flipping in these steps is $O(n^{1/2})$. This implies that the number of bad triangles is decreased by only $O(n^{1/2})$. □

Claim 7.3.6 *Let $b(y) = \Theta(n)$ and $\mathrm{con}_G(y) = 1$. In a phase of length $n^{5/2}$, a search point z where $b(z) = \Theta(n)$, $\mathrm{con}_G(z) = 1$, and $T(z)$ is a tree is produced.*

Proof: Only search points x describing connected graphs are accepted, in particular, $d(x) = 0$. Let z be the first search point where $T(z)$ is a tree. Then $\mathrm{con}_G(z) = 1$ and we have to prove that $b(z) = \Theta(n)$ and that z is produced within $n^{5/2}$ steps. A 1-step can be accepted only if it turns a complete triangle into a good or bad triangle. Such a step is accepted if no other edge flips. Moreover, $c(x)$ cannot be increased. In order to increase $c(x)$ it is necessary to add the missing edge to a good or bad triangle. To compensate this weight increase, we have to eliminate an edge of a complete triangle. Remember that we have no k-steps for $k \geq 3$. If $c(x) = l$, the probability of decreasing the c-value is at least $3l/(em)$ and the expected time to eliminate all complete triangles is $O(m \log n) = O(n^2 \log n)$. Hence, $n^{5/2}$ steps are sufficient to create z. The number of bad triangles can be decreased only in the $O(n^{1/2})$ 2-steps implying that $b(z) = \Theta(n)$. □

Claim 7.3.7 *Let $b(z) = \Theta(n)$, $\mathrm{con}_G(z) = 1$, and $T(z)$ be a tree. The expected time to find a minimum spanning tree is $\Omega(n^4 \log n)$.*

Proof: First, we assume that only 2-steps change the number of bad triangles. Later, we complete the arguments. The expected waiting time for a 2-step flipping those two edges of a bad triangle that turn it into a good one equals $\Theta(n^4)$. The expected time to decrease the number of bad triangles from b to $b-1$ equals $\Theta(n^4/b)$. Since b has to be decreased from $\Theta(n)$ to 0, we obtain an expected waiting time of

$$\Theta(n^4 \cdot \sum_{1 \leq b \leq \Theta(n)} (1/b)) = \Theta(n^4 \log n). \qquad (*)$$

Similarly to the proof of the coupon collector's theorem (see e. g. Motwani and Raghavan (1995)) we obtain that the optimization step if only 2-steps can be accepted equals $\Theta(n^4 \log n)$ with probability $1 - o(1)$. Hence, it is sufficient to limit the influence of all k-steps, $k \neq 2$, within a time period of $\alpha n^4 \log n$ for some constant $\alpha > 0$. Again with probability $1 - o(1)$, the number of 4-steps is $O(\log n)$ and there are no k-steps for $k \geq 5$. The 4-steps can decrease the number of bad triangles by at most $O(\log n)$. Because of the weight increase, a k-step, $k \leq 4$, can be accepted only if it eliminates at least $\lceil k/2 \rceil$ triangle edges. Moreover, it is not possible to disconnect a good or a bad triangle. Hence, a 4-step cannot create a complete triangle. As long as there is no complete triangle, a 3-step or a 1-step has to disconnect a triangle and is not accepted. A 2-step can only be accepted if it changes a bad triangle into a good one. Hence, no complete triangles are created. The 4-steps eliminate $O(\log n)$ terms of the sum in $(*)$. The largest terms are those for the smallest values of b. We only have to substract a term of $O(n^4 \log \log n) = o(n^4 \log n)$ from the bound $\Theta(n^4 \log n)$ and this proves the claim. □

This completes the proof since the sum of all failure probabilities is $o(1)$. □

In the following, we prove an upper bound of size $O(m^2(\log n + \log w_{\max}))$ on the expected optimization time for arbitrary graphs using the method of the expected multiplicative weight decrease (see Section 5.3). This bound relies on the properties of minimum spanning trees which we have stated in Section 7.2 and is $O(m^2 \log n)$ as long as $w_{\max}$ is polynomially bounded. But it is always polynomially bounded with respect to the bit length of the input. Theorem 7.3.3 shows that the bound is optimal.

Theorem 7.3.8 *The expected time until RLS_b or (1+1) EA_b working on the fitness function w constructs a minimum spanning tree is bounded by $O(m^2(\log n + \log w_{\max}))$.*

Proof: By Lemmas 7.3.1 and 7.3.2, it is sufficient to investigate the search process after having found a search point s describing a spanning T. Then, by Lemma 7.2.3, there always exists a set of n 2-bit flips whose average weight decrease is at least $(w(s) - w_{\mathrm{opt}})/n$. The choice of such a 2-bit flip is called a "good step". The probability of performing a good step equals $\Theta(n/m^2)$ and each of the good steps is chosen with the same probability. A

good step decreases the difference between the weight of the current spanning tree and w_{opt} on average by a factor not larger than $1 - 1/n$. This holds independently from previous good steps. Hence, after N good steps, the expected difference of the weight of T and w_{opt} is bounded above by $(1 - 1/n)^N \cdot (w(s) - w_{\text{opt}})$. Since $w(s) \leq (n-1) \cdot w_{\max}$ and $w_{\text{opt}} \geq 0$, we obtain the upper bound $(1 - 1/n)^N \cdot D$, where $D := n \cdot w_{\max}$.

If $N := \lceil (\ln 2) \cdot n \cdot (\log D + 1) \rceil$, this bound is at most $\frac{1}{2}$. Since the difference is not negative, by Markov's inequality, the probability that the bound is less than 1 is at least $1/2$. The difference is an integer implying that the probability of having found a minimum spanning tree is at least $1/2$. Repeating the same arguments, the expected number of good steps until a minimum spanning tree is found is bounded by $2N = O(n \log D) = O(n(\log n + \log w_{\max}))$.

By our construction, there are always exactly n good 2-bit flips. Therefore, the probability of a good step does not depend on the current search point. Hence, the expected time until r steps are good equals $\Theta(rm^2/n)$. Altogether, the expected optimization time is bounded by

$$O(Nm^2/n) = O(m^2(\log n + \log w_{\max})). \qquad \square$$

Applying Lemma 7.2.4 instead of Lemma 7.2.3, it is not too difficult to obtain the same upper bound for the fitness function w'. The main difference is that a good 1-bit flip has a larger probability than a good 2-bit flip.

Theorem 7.3.9 *The expected time until RLS_b or (1+1) EA_b working on the fitness function w' constructs a minimum spanning tree is bounded by $O(m^2(\log n + \log w_{\max}))$.*

Proof: By Lemma 7.3.1, it is sufficient to analyze the phase after having constructed a connected graph. We apply Lemma 7.2.4. The total weight decrease of the chosen 1-bit flips and 2-bit flips is at least $w(s) - w_{\text{opt}}$ if s is the current search point. If the total weight decrease of the 1-bit flips is larger than the total weight decrease of the chosen 2-bit flips, the step is called a 1-step. Otherwise, it is called a 2-step.

If more than half of the steps are 2-steps, we adapt the proof of Theorem 7.3.8 with $N' := 2N$ since we guarantee only an expected weight decrease by a factor of $1 - 1/(2n)$. Otherwise, we consider the good 1-steps which have an expected weight decrease by a factor of $1 - 1/(2m')$ for $m' = m - (n-1)$. Choosing $M := \lceil 2 \cdot (\ln 2) \cdot m' \cdot (\log D + 1) \rceil$, we can apply the proof technique of Theorem 7.3.8 where M takes the role of N. The probability of performing a good 1-bit flip equals $\Theta(m'/m)$. In this case, we obtain the bound

$$O(Mm/m') = O(m(\log n + \log w_{\max}))$$

for the expected number of steps which is even smaller than the proposed bound. $\square$

7.4 Generalizations

Theorems 7.3.3, 7.3.8, and 7.3.9 contain matching upper and lower bounds for RLS_b and (1+1) EA_b with respect to the fitness functions w and w'. The bounds are worst-case bounds and one can hope that the algorithms are more efficient for many graphs. Here we discuss what can be gained by other randomized search heuristics.

First, we introduce more problem-specific mutation operators. It is easy to construct spanning trees. Afterwards, it is good to create children with the same number of edges. The new mutation operators are:

- If RLS_b flips two bits, it chooses randomly a 0-bit and randomly a 1-bit.
- If s contains k 1-bits, (1+1) EA_b flips each 1-bit with probability $1/k$ and each 0-bit with probability $1/(m-k)$.

For spanning trees, the probability of a specific edge exchange is increased from $\Theta(1/m^2)$ to $\Theta(1/(n(m-n+1)))$. The following result can be obtained by adjusting the proofs of the previous section to the modified mutation operators.

Theorem 7.4.1 *For the modified mutation operators, the bounds of Theorems 7.3.3, 7.3.8, and 7.3.9 can be replaced by bounds of size* $\Theta(mn \log n)$ *and* $O(mn(\log n + \log w_{\max}))$ *respectively.*

Using larger populations, we have to pay for improving all members of the population. This holds at least if we guarantee a large diversity in the population. The lower bound of Theorem 7.3.3 holds with overwhelming probability. Hence, we do not expect that large populations help. The analysis in the proof of Theorems 7.3.8 and 7.3.9 is quite precise in most aspects. There is only one essential exception. We know that the weight distance to w_{opt} is decreased on average by a factor of at most $1-1/n$ and we work under the pessimistic assumption that this factor equals $1-1/n$. For large populations or multi-starts the probability of having sometimes much larger improvements may increase for many graphs.

It is more interesting to "parallelize" the algorithms by producing more children in parallel. The algorithm (1+λ) EA_b differs from (1+1) EA_b by producing in each iteration independently λ children from the single individual of the current population. The selection procedure selects an individual with the smallest w-value (or w'-value) among the parent and its children. In a similar way, we obtain λ-PRLS_b (parallel RLS_b) from RLS_b. In the proofs of Theorem 7.3.8 and Theorem 7.3.9 we have seen that the probability of a good step is $\Theta(n/m^2)$. Choosing $\lambda = \lceil m^2/n \rceil$, this probability is increased to a positive constant. We have seen that the expected number of good steps is bounded by $O(n(\log n + \log w_{\max}))$. This leads to the following result.

Theorem 7.4.2 *The expected number of generations until λ-PRLS$_b$ or the (1+λ) EA$_b$ with $\lambda := \lceil m^2/n \rceil$ children constructs a minimum spanning tree is $O(n(\log n + \log w_{\max}))$. This holds for the fitness functions w and w'.*

If we use the modified mutation operator defined above, the probability of a good step is $O(1/m)$ and we obtain the same bound on the expected number of generations as in Theorem 7.4.2 already for $\lambda := m$.

Jansen and Sudholt (2005) have analyzed this mutation operator later in greater detail. One important result of their work is that simple but not trivial pseudo-boolean functions can be optimized by evolutionary algorithms in time $O(n)$ which breaks for the first time the $\Theta(n \log n)$ bound that is often the result of the analysis on simple pseudo-boolean functions.

Crossover operators are considered as important in evolutionary computation. But one-point crossover or two-point crossover are not appropriate for edge set representations. It is not possible to build blocks of all edges adjacent to a vertex. For uniform crossover, it is very likely to create graphs which are not spanning trees. Hence, only problem-specific crossover operators seem to be useful. Such operators are described by Raidl and Julstrom (2003). It is difficult to analyze heuristics with these crossover operators and no results with respect to the runtime of such algorithms have been obtained up to now.

7.5 Conclusions

The minimum spanning tree problem is one of the fundamental problems which are efficiently solvable. Several important variants of this problem are difficult, and evolutionary algorithms have a good chance to be competitive on these problems. As a first step toward the analysis of evolutionary algorithms on these problems, randomized local search and simple evolutionary algorithms have been analyzed on the basic minimum spanning tree problem. The asymptotic worst-case (with respect to the problem instance) expected optimization time has been obtained exactly. The analysis is based on the investigation of the expected multiplicative weight decrease (with respect to the difference of the weight of the current graph and the weight of a minimum spanning tree).

A conference version that contains the results of this chapter has been published in the Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2004 (see Neumann and Wegener (2004)). A journal version of this chapter has been submitted for publication.

Chapter 8

A Simple ACO Algorithm

In this chapter, we focus on another kind of randomized search heuristics, namely ant colony optimization (ACO). Like EAs, these heuristics imitate optimization processes from nature, in this case the search of an ant colony for a common source of food. Solving problems by ACO techniques has become quite popular in recent years. From a theoretical point of view, there are so far no results that provide estimates of the runtime of ACO algorithms. Despite interesting theoretical investigations of models and dynamics of ACO algorithms (Dorigo and Blum (2005)), convergence results are so far the only results related to their runtimes. Dorigo and Blum (2005) explicitly formulate the open problem to determine the runtime of ACO algorithms on simple problems in a similar fashion to what has been done for EAs.

We solve this problem, starting the analysis of ACO algorithms with respect to their expected runtimes and success probability after a specific number of steps. Randomized local search, Simulated annealing, and the Metropolis algorithm (see Section 3.3), search more or less locally, and runtime bounds are often obtained by considering the neighborhood structure of the considered problem. The same holds for simple evolutionary algorithms. Considering ACO algorithms, this is different as search points are obtained by random walks of ants on a construction graph. The traversal of an ant on this directed graph is determined by values on the edges which are called pheromone values (see Section 3.2). Larger pheromone values correspond to a higher probability of traversing a certain edge, where the choice of an edge usually fixes a parameter in the current search space. The pheromone values are updated if a good solution has been constructed in this random walk. This update depends on the traversal of the ant and a so-called evaporation factor ρ.

The choice of ρ seems to be a crucial parameter in an ACO algorithm. Using a large value of ρ, the last accepted solution changes the pheromone values by a large amount such that there is a large probability of producing this solution in the next step. In contrast to this, the use of a small evaporation factor leads to a small effect of the last accepted solution such that an improvement may be hard to find in the next step. We consider the 1-ANT algorithm introduced in Section 4.1.3 and show that it behaves for large values of ρ

as the simple evolutionary algorithm (1+1) EA_b. This algorithm has been studied extensively with respect to its runtime on classes of pseudo-boolean functions (see, e. g. Droste, Jansen, and Wegener (2002)) as well as on combinatorial optimization problems. The list of problems where runtime bounds have been obtained include some of the best-known polynomially solvable problems such as maximum matchings (Giel and Wegener (2003)) and minimum spanning trees (see Chapter 7). In the case of NP-hard problems, one is usually interested in good approximations of optimal solutions. Witt (2005) has presented a worst-case and average-case analysis of (1+1) EA_b for the partition problem, which is one of the first results on NP-hard problems. All these results immediately transfer to 1-ANT with large ρ.

After having obtained these general results, we consider the effect of the evaporation factor ρ on the runtime of 1-ANT in detail. This analysis requires new techniques since it is the first one of its kind. We examine the simplest non-trivial pseudo-boolean function called OneMax and show that small values of ρ with high probability lead to an exponential optimization time even for this simple function. In addition, we examine for which choices of ρ the optimization time with high probability is still upper bounded by a small polynomial. To achieve these bounds, we consider the expected function value for the algorithm in the next step. It turns out that larger values of ρ change the pheromone values on the edges such that the expected value in the next step is determined by the function value of the best seen solution. Using results obtained by Hoeffding (1956) on the sum of independent Poisson trails, we show that an improvement will be achieved after an expected polynomial number of steps. In the case of small ρ, achieving an improvement does not increase the expected value in the next step that much. Here exponential lower bounds are obtained by showing that there is a large gap between the expected value and the best-so-far function value. Both the proof of the upper and the lower runtime bound contain new analytical tools to lower bound the tail of a sum of independent trials with different success probabilities. The new tools may be of independent interest in other probabilistic analyses.

We investigate the relation of 1-ANT to (1+1) EA_b in Section 8.1 and transfer the results on this EA to our algorithm. After that we investigate the behavior of 1-ANT on the function OneMax in greater detail. In Section 8.2 we give investigate the choice of ρ that leads to an exponential runtime and show in Section 8.3 that a slightly larger value of ρ reduces the runtime from exponential to polynomial.

8.1 1-ANT and (1+1) EA_b

We consider the relation between 1-ANT and (1+1) EA_b and show that they behave for large values of ρ identical. (1+1) EA_b has extensively been studied with respect to its runtime distribution. We consider the optimization of pseudo-boolean goal functions $f\colon \{0,1\}^n \to \mathbb{R}$ for $n \geq 3$. 1-ANT produces solutions by random walks on the construction graph C_{bool} shown in Figure 4.1. The random walk is performed using the construction

procedure Construct(C, τ) described in Section 4.1.3. The result of this procedure is a solution x together with the path $P(x)$ that has been used to construct this solution.

The algorithm (1+1) EA_b chooses the first solution x uniformly at random from $\{0,1\}^n$. After that it produces in each iteration a new solution x' from a currently best solution x by flipping each bit of x with probability $1/n$. Hence, the probability of producing a certain solution x' with Hamming distance $H(x,x')$ to x is $(1/n)^{H(x,x')} \cdot (1-1/n)^{n-H(x,x')}$.

In the following, we consider 1-ANT with values of ρ at least $\frac{n-2}{3n-2}$, which is for large n approximately $1/3$. In this case, we show that 1-ANT behaves as (1+1) EA_b on each function. This also means that 1-ANT has the same expected optimization time as (1+1) EA_b on each function.

Theorem 8.1.1 *Choosing $\rho \geq (n-2)/(3n-2)$, 1-ANT has the same runtime distribution as (1+1) EA_b on each pseudo-boolean function.*

Proof: In the initialization step of (1+1) EA_b, a bitstring is chosen uniformly at random, which means that $\text{Prob}(x_i = 1) = \text{Prob}(x_i = 0) = 1/2$ for all i, $1 \leq i \leq n$. As $\tau_{(u,v)} = 1/(4n)$ holds for each edge $(u,v) \in E$, the probability to choose the edge (v_{3i}, v_{3i+1}) equals the probability of choosing the edge (v_{3i}, v_{3i+2}) at vertex v_{3i}, $0 \leq i \leq n-1$, and is $1/2$. Hence, the 1-ANT chooses the first solution uniformly at random from the search space $\{0,1\}^n$ as (1+1) EA_b.

Assume that the up to now best solution constructed by 1-ANT is x^*. This implies that the edges of the construction graph corresponding to this solution have been updated in the last update operation. Before the update, the value $\tau_{(u,v)}$ of each edge $(u,v) \in P(x^*)$ was at least $\frac{1}{2n^2}$ and the value $\tau_{(u,v)}$ of edges $(u,v) \notin P(x^*)$ was at most $\frac{n-1}{2n^2}$.

We inspect the case of an edge $(u,v) \in P(x^*)$ in greater detail and consider the function

$$h(\rho) := \frac{(1-\rho)\cdot\tau_{(u,v)}+\rho}{1-\rho+2n\rho} \geq \frac{(1-\rho)\cdot\frac{1}{2n^2}+\rho}{1-\rho+2n\rho} = \frac{1}{2n^2}\cdot\frac{1+(2n^2-1)\rho}{1+(2n-1)\rho} =: h'(\rho).$$

For each fixed $n \geq 1$, h' is a non-decreasing function. Using $\rho \geq (n-2)/(3n-2)$, we get

$$h(\rho) \geq \frac{1+(2n^2-1)\frac{n-2}{3n-2}}{2n^2+(4n^3-2n^2)\frac{n-2}{3n-2}} = \frac{2n^3-4n^2+2n}{4n^4-4n^3} = \frac{n-1}{2n^2}.$$

Hence, the pheromone value of each edge $(u,v) \in P(x^*)$ is $\frac{n-1}{2n^2}$ after the update. The pheromone value of each edge $(u,v) \notin P(x^*)$ is $\frac{1}{2n^2}$ as the sum of the pheromone values of two complementary edges is $\frac{1}{2n}$. After this update, the probability to choose in the next solution x the bit $x_i = x_i^*$ is $\frac{2n(n-1)}{2n^2} = 1-\frac{1}{n}$ and the probability to choose $x_i = 1-x_i^*$ is $\frac{2n}{2n^2} = 1/n$. Hence the probability to produce a specific solution x that has Hamming distance $H(x^*,x)$ to x^* is $(1/n)^{H(x^*,x)} \cdot (1-1/n)^{n-H(x^*,x)}$ which completes the proof. □

8.2 Exponential lower bounds for OneMax

In the following, we inspect the choice of ρ in greater detail for a simple pseudo-boolean function called OneMax and defined by $\textsc{OneMax}(x) = \sum_{i=1}^{n} x_i$. This is the simplest non-trivial pseudo-boolean function that can be considered and analyses of ACO algorithms for such simple functions are explicity demanded by Dorigo and Blum (2005). Note that due to results on the (1+1) EA_b by Droste, Jansen, and Wegener (2002), the expected optimization time of 1-ANT is $O(n \log n)$ on each linear function if $\rho \geq (n-2)/(3n-2)$ holds.

We prepare ourselves by considering the effects of pheromone updates for a solution x^* in greater detail. Let $\tau(e)$ and $\tau'(e)$ be the pheromone values on edge e before respectively after the update. If $e \in P(x^*)$, $\tau'(e) \geq \tau(e)$ and $\tau'(e) \leq \tau(e)$ otherwise. The amount by which the pheromone value is increased on a 1-edge equals the amount the pheromone value is decreased on the complementary 0-edge. However, the change of a pheromone value depends on the previous value on the edge. In the following lemma, we bound the relative change of pheromone values. We call an edge *saturated* iff its pheromone value is either $\frac{1}{2n^2}$ or $\frac{n-1}{2n^2}$.

Lemma 8.2.1 *Let e_1 and e_2 be two edges of C_{bool} and let τ_1 respectively τ_2 be their current pheromone values in 1-ANT. Let τ_1' respectively τ_2' be their updated pheromone values for the next accepted solution x. If $e_1, e_2 \in P(x^*)$ and none of the edges is saturated before or after the update, then $|(\tau_1' - \tau_1) - (\tau_2' - \tau_2)q| \leq \rho|\tau_1 - \tau_2|$.*

Proof: W. l. o. g., $\tau_2 \geq \tau_1$. Since $e_1, e_2 \in P(x^*)$ and no edge is saturated,

$$\tau_1' \;=\; \frac{(1-\rho)\tau_1 + \rho}{1 - \rho + 2n\rho} \quad \text{and} \quad \tau_2' \;=\; \frac{(1-\rho)\tau_2 + \rho}{1 - \rho + 2n\rho}.$$

This implies

$$(\tau_1' - \tau_1) - (\tau_2' - \tau_2) \;=\; \frac{\rho - \tau_1 2n\rho - (\rho - \tau_2 2n\rho)}{1 - \rho + 2n\rho} \;\geq\; 0.$$

Second, since the denominator is at least 1, we obtain

$$\tau_1' - \tau_2' \;\leq\; \rho(\tau_2 - \tau_1) + (\tau_1 - \tau_2) \quad \Rightarrow \quad (\tau_1' - \tau_1) - (\tau_2' - \tau_2) \;\leq\; \rho|\tau_1 - \tau_2|.$$

Taking the absolute value of $(\tau_1' - \tau_1) - (\tau_2' - \tau_2)$, the claim follows. □

In the following, we will figure out which values of ρ lead to efficient runtimes of 1-ANT and which do not. Intuitively, $1/n$ is a threshold value for ρ since the denominator of the normalization factor $1 - \rho + 2n\rho$ diverges for $\rho = \omega(1/n)$ and is $1 - \rho - o(1)$ for $\rho = o(1/n)$. We will make precise that the behavior of 1-ANT on OneMax changes drastically when ρ is asymptotically smaller respectively larger than $1/n$.

Choosing $\rho = 0$, the pheromone value on each edge is $1/(4n)$ at each time step. This implies that the expected optimization time of the 1-ANT on OneMax is 2^n as each solution is chosen uniformly at random from $\{0,1\}^n$. In the following, we show that the optimization time with overwhelming proability is still exponential if ρ is convergent to 0 only polynomially fast.

Assume that the currently best solution x^* has value k. Then the following lemma gives a lower bound on the probability of overshooting k by a certain amount in the next accepted step.

Lemma 8.2.2 *Let $X_1, \ldots, X_n \in \{0,1\}$ be independent Poisson trials with success probabilities p_i, $1 \leq i \leq n$. Let $X := X_1 + \cdots + X_n$, $\mu := E(X) = p_1 + \cdots + p_n$ and $\sigma := \sqrt{\mathrm{Var}(X)}$. For any $0 \leq k \leq n - \sigma$, let $\gamma_k = \max\{2, (k-\mu)/\sigma\}$. If $\sigma = \omega(1)$ then $\mathrm{Prob}(X \geq k + \sigma/\gamma_k \mid X \geq k) = \Omega(1)$.*

Proof: Since the X_i are bounded and σ diverges, Lindeberg's generalization of the Central Limit Theorem (Feller (1971), Chapter VIII.4) holds s. t. the distribution of X converges to a Normal distribution with expectation μ and variance σ^2. We use approximations of the Normal distribution (with the common notion $\Phi(x)$ for its cumulative distribution function) and distinguish two cases.

If 2 maximizes γ_k, we even show $\tilde{p}_k := \mathrm{Prob}(X \geq k + \sigma/\gamma_k) = \Omega(1)$. Let $\tilde{d}_k := (k + \sigma/\gamma_k - \mu)/\sigma$ be the normalized deviation from the expectation. Since by our assumptions $(k-\mu)/\sigma \leq 2$, we obtain $\tilde{d}_k = O(1)$. The Central Limit Theorem implies $\tilde{p}_k = (1 \pm o(1))(1 - \Phi(\tilde{d}_k)) = \Omega(1)$.

Now let $\gamma_k > 2$. Let $p_k := \mathrm{Prob}(X \geq k)$, $d_k := (k-\mu)/\sigma$, and let $\tilde{p}_k$ and $\tilde{d}_k$ as above. By our assumptions, $2 \leq d_k \leq \tilde{d}_k \leq d_k + 1/d_k$. We have to bound $\tilde{p}_k/p_k$ from below. We reuse the Central Limit Theorem and employ the inequalities

$$\left(\frac{1}{x} - \frac{1}{x^3}\right) \cdot \frac{1}{\sqrt{2\pi}} \cdot e^{-x^2/2} \quad < \quad 1 - \Phi(x) \quad < \quad \frac{1}{x} \cdot \frac{1}{\sqrt{2\pi}} \cdot e^{-x^2/2}$$

(see Feller (1968), Chapter VII.1). Hence,

$$\frac{\tilde{p}_k}{p_k} \geq \frac{1 - o(1)}{1 + o(1)} \cdot \left(\frac{d_k}{\tilde{d}_k} - \frac{d_k}{(\tilde{d}_k)^3}\right) \cdot e^{-(1/2)((\tilde{d}_k)^2 - (d_k)^2)}.$$

The first fraction and the ()-term are $\Omega(1)$. Finally, the e-term is $\Omega(1)$ since $(\tilde{d}_k)^2 - (d_k)^2 \leq (d_k + 1/d_k)^2 - (d_k)^2 \leq 2 + 1/(d_k)^2 \leq 3$. □

Using this lemma, we are able to prove an exponential lower bound on the runtime of 1-ANT on OneMax. In order to show that the success probability in an exponential number of steps is still exponentially small, we assume that $\rho = O(n^{-1-\epsilon})$ for some constant $\epsilon > 0$.

Theorem 8.2.3 *Let $\rho = O(n^{-1-\epsilon})$ for some constant $\epsilon > 0$. Then the optimization time of 1-ANT on* OneMax *is $2^{\Omega(n^{\epsilon/3})}$ with probability $1 - 2^{-\Omega(n^{\epsilon/3})}$.*

Proof: The main idea is to keep track of the so-called 1-*potential*, defined as the sum of pheromone values on 1-edges. Note that the 1-potential multiplied by n equals the expected OneMax-value of the next constructed solution x. If the 1-potential is bounded above by $1/2+O(1/\sqrt{n})$, Chernoff bounds yield that the probability of $\text{OneMax}(x) \geq n/2+n^{1/2+\epsilon/3}$ is bounded above by $2^{-\Omega(n^{\epsilon/3})}$. We will show that with overwhelming probability, the 1-potential is bounded as suggested as long as the OneMax-value of the so far best solution is bounded above by $n/2 + n^{1/2+\epsilon/3}$.

Starting with initialization, we consider a phase of length $s := \lfloor 2^{cn^{\epsilon/3}} \rfloor$ for some constant c to be chosen later and show that the success probability in the phase is $2^{-\Omega(n^{\epsilon/3})}$. A main task is to bound the number of successful steps of the phase, i. e., of steps where the new solution is accepted and a pheromone update occurs. In a success with OneMax-value $n/2+i$, $n+2i$ pheromone values on 1-edges are increased and $n-2i$ are decreased. Suppose all pheromone values are $1/(4n) \pm o(1/n)$ in the phase. Then Lemma 8.2.1 yields that the 1-potential is changed by at most $4i(1 \pm o(1))\rho$ due to the considered success. Hence, if the best solution always had OneMax-value at most $n/2 + n^{1/2+\epsilon/3}$, the total change of the 1-potential due to at most $O(n^{2\epsilon/3})$ successes would be at most

$$O(n^{2\epsilon/3}) \cdot 4n^{1/2+\epsilon/3} \cdot (1 \pm o(1))\rho \;=\; O(n^{1/2+\epsilon}) \cdot O(1/n^{1+\epsilon}) \;=\; O(1/n^{1/2})$$

by our assumption on ρ. This would prove the theorem since the initial 1-potential is $1/2$.

Under the assumption on the pheromone values, we want to show that with probability $1-2^{-\Omega(n^{\epsilon/3})}$, at most $c'n^{2\epsilon/3}$ successes occur in the phase, where c' is an appropriate constant. We already know that then the probability of a success with value at least $n/2 + n^{1/2+\epsilon/3}$ is $2^{-\Omega(n^{\epsilon/3})}$ in each step of the phase. If c is chosen small enough, this probability is $2^{-\Omega(n^{\epsilon/3})}$ for the whole phase. Moreover, the initial value is at least $n/2 - n^{1/2+\epsilon/3}$ with probability $1 - 2^{-\Omega(n^{\epsilon/3})}$.

Let the so far best value be k. We apply Lemma 8.2.2 with respect to the expected OneMax-value μ of the next constructed solution. Note that $k - \mu = O(n^{1/2+\epsilon/3})$ holds at each time step we consider. Moreover, $p_i = 1/2 \pm o(1)$ is assumed to hold for all bits, implying $\sigma = \Theta(n^{1/2})$. Hence, with probability $\Omega(1)$ the next success leads to a value at least $k+\Omega(n^{1/2-\epsilon/3})$. Using Chernoff bounds, with probability $1-2^{-\Omega(n^{\epsilon/3})}$, $c'n^{2\epsilon/3}$ successes increase the OneMax-value by at least $c''n^{1/2+\epsilon/3}$, where c'' is an appropriate constant.

We still have to show the statement on the pheromone values. This is not too difficult for our choice of ρ if the number of successes is bounded by $O(n^{2\epsilon/3})$. Then the total change of pheromone on any fixed edge is bounded above by

$$\rho \cdot O(n^{2\epsilon/3}) \;=\; O(n^{-1-\epsilon}) \cdot O(n^{2\epsilon/3}) \;=\; o(1/n)$$

with probability $1 - 2^{-\Omega(n^{\epsilon/3})}$. Since the number of edges is bounded by $4n$, this holds also for all edges together. Since the sum of all failure probabilities is $2^{-\Omega(n^{\epsilon/3})}$, this completes the proof. □

8.3 Polynomial upper bounds for OneMax

In the following, we consider for which values of ρ the optimization time of 1-ANT on OneMax with high probability is still polynomial. We will show that the function value of the last accepted solution determines the expected value of the next solution almost exactly if $\rho = \Omega(n^{-1+\epsilon})$, $\epsilon > 0$ an arbitrary constant. To determine the expected time to reach an improvement, we give a lower bound on the probability of overshooting the expected value by at least a small amount.

Lemma 8.3.1 *Let $X_1, \ldots, X_k \in \{0,1\}$, $k \leq n$, be independent Poisson trials with success probabilities $p_i \in [1/n, 1-1/n]$, $1 \leq i \leq k$. Let $X := X_1 + \cdots + X_k$ and $\mu := E(X) = p_1 + \cdots + p_k$. If $\mu \leq k-1$ then* $\text{Prob}(X \geq \mu + 1/2) = \Omega(1/n)$.

Proof: It follows from the work by Hoeffding (1956) that $\text{Prob}(X \geq \mu + 1/2)$ is minimized if the p_i take on at most 3 different values, only one of which is distinct from $1/n$ and $1-1/n$. (See Lemma B.1.4 in Appendix B.)

Let n_ℓ be the number of p_i that are $1/n$, n_h be the number that are $1-1/n$ and n_a be the number that take a different value a, $1/n < a < 1-1/n$. The random variables belonging to each of the three sets are called ℓ-variables, h-variables and a-variables, respectively. Let X_ℓ, X_h and X_a be the sums of the variables from these sets, i. e., $X = X_\ell + X_h + X_a$, and let $\mu_\ell = n_\ell/n$, $\mu_h = n_h(1-1/n)$ and $\mu_a = n_a a$ be the corresponding expectations. In the following arguments we also cover the case that up to two sets are empty.

It always holds that $X_h = n_h \geq \mu_h$ with probability $(1-1/n)^{n_h} = \Omega(1)$. We distinguish several cases according to the variables from the other two sets. Since by assumption $\mu \leq k-1$, the simple case $n_a = n_\ell = 0$ is only possible if $k = n$ since otherwise $\mu = k - k/n > k-1$. In this simple case, however, $X = n = \mu + 1$ holds with probability $\Omega(1)$. In the following, we therefore assume that $n_a > 0$ or $n_\ell > 0$ (or both). First we concentrate on the most complicated case that $n_\ell \neq 0 \neq n_a$, implying $\mu_\ell \neq 0 \neq \mu_a$. If $0 < \mu_\ell \leq 1/4$ and $0 < \mu_a \leq 1/4$, we exploit that $X_\ell \geq 1$ with probability at least $1/n$. Hence $X_h + X_\ell \geq n_h + 1 \geq \mu_h + (\mu_\ell + 3/4) \geq \mu + 1/2$ with probability $\Omega(1/n)$.

Now let $\mu_a > 1/4$ and $0 < \mu_\ell \leq 1/4$. We distinguish four cases depending on n_a, μ_a and $\sigma_a = \sqrt{n_a a(1-a)}$. In all cases, we exploit that $X_\ell \geq 1 \geq \mu_\ell + 3/4$ with probability $\Omega(1/n)$.

Case 1: $n_a = O(1)$. Since $\mu_a \geq 1/4$, $a = \Omega(1)$. Hence, we have $X_a = n_a \geq \mu_a$ with probability $\Omega(1)$, implying $X = X_h + X_a + X_\ell \geq \mu_h + \mu_a + \mu_\ell + 3/4 = \mu + 3/4$ with probability $\Omega(1/n)$.

Case 2: $n_a \to \infty$ and $\mu_a = O(1)$. Hence, X_a can be approximated by means of the Poisson distribution with parameter μ_a, implying $X_a \geq \mu_a$ with probability at least $(1-o(1)) \cdot e^{-\mu_a} (\mu_a)^{\lceil \mu_a \rceil} / (\lceil \mu_a \rceil)! = \Omega(1)$. Hence, we conclude as in Case 1 that $X \geq \mu + 3/4$ with probability $\Omega(1/n)$.

Case 3: $\mu_a \to \infty$ (implying $n_a \to \infty$) and $\sigma_a \to \infty$. Using the Central Limit Theorem (Feller (1968)), we approximate X_a by a Normal distribution, implying $X_a \geq \mu_a$ with probability $\Omega(1)$. We go on as in Case 1.

Case 4: $\mu_a \to \infty$ and $\sigma_a = O(1)$. Since $\sigma_a^2 = \mu_a(1-a) = O(1)$ implies $a \geq 1/2 - o(1)$, we obtain $1 - a = O(1/\mu_a) = O(1/n_a)$. Hence, $X_a = n_a \geq \mu_a$ with probability at least $(1 - O(1/n_a))^{n_a} = \Omega(1)$. We go on as in Case 1.

The case that $0 < \mu_a \leq 1/4$ and $\mu_\ell > 1/4$ can be handled by an analogous case distinction according to n_ℓ and μ_ℓ. Here some cases are even impossible.

We still have to study the situation that $\mu_a > 1/4$ and $\mu_\ell > 1/4$. Then we examine to which of the four cases the a-variables and ℓ-variables belong. If both lead to one of the Cases 1 or 4, we even obtain $X = k \geq \mu + 1$ with probability $\Omega(1)$. Otherwise, at least one of the two sets of variables leads to Case 2 or 3. W. l. o. g., let this be the a-variables. Then $n_a \to \infty$ and $n_a - \mu_a \to \infty$ and both the approximation by the Poisson and the Normal distribution can be adapted in a straightforward manner to show that even $X_a \geq \mu_a + 1/2$ still holds with probability $\Omega(1)$.

Finally, we have to consider the case that $n_\ell = 0 \neq n_a$ or $n_a = 0 \neq n_\ell$. It suffices to study the case $n_a \neq 0 = n_\ell$. Considering the above four cases and the extra case $\mu_a \leq 1/4$, the lemma follows by the same arguments as before. □

Using Lemma 8.3.1 we will show that with probabilty close to 1 an optimal solution is produced within $O(n^2)$ iterations of 1-ANT if ρ is slightly larger than $1/n$.

Theorem 8.3.2 *Choosing $\rho = \Omega(n^{-1+\epsilon})$, $\epsilon > 0$ a constant, the optimization time of 1-ANT on* OneMax *is $O(n^2)$ with probability $1 - 2^{-\Omega(n^{\epsilon/2})}$.*

Proof: We assume $\rho \leq 1/2$ since the result follows from Theorem 8.1.1 otherwise. In contrast to previous definitions, an edge is called saturated if its pheromone value is $\frac{n-1}{2n^2}$ and called unsaturated otherwise. Let x^* be a newly accepted solution and denote by $\mathcal{S}$ the set of saturated 1-edges and by $\mathcal{U}$ the set of unsaturated 1-edges after the pheromone update. Let $k = \text{OneMax}(x^*)$ and decompose k according to $k = k_\text{s} + k_\text{u}$, where k_s denotes the number of ones in x^* whose corresponding 1-edges belong to $\mathcal{S}$ and k_u to the number of ones in x^* whose 1-edges belong to $\mathcal{U}$. The probability that the edges of $\mathcal{S}$ contribute at least k_s to the next (not necessarily accepted) solution x is at least $(1-1/n)^{k_\text{s}} = \Omega(1)$.

Consider the potential P of all edges of $\mathcal{U}$ before x^* updates the pheromone values. Let $\mu = Pn$ be the expected OneMax-value w. r. t. these edges before the update. Depending on P and k_u, we compute $P^*(\rho)$, the new 1-potential on these edges:

$$P^*(\rho) = \frac{(1-\rho)P + 2k_\text{u}\rho}{(1-\rho) + 2n\rho}.$$

We denote by $\mu^* = P^*(\rho) \cdot n$ the expected OneMax-value w. r. t. to edges of $\mathcal{U}$ after the update has occured. Under certain assumptions, we will prove that with probability $1 - 2^{-\Omega(n^\epsilon)}$, $\mu^* + 1/2 > k_\text{u}$. Since k_u is an integer, Lemma 8.3.1 shows that the probability of producing in the next solution x at least $\lceil \mu^* + 1/2 \rceil \geq k_\text{u} + 1$ ones by the mentioned edges is at least $\Omega(1/n)$. Considering the difference between μ^* and k_u, we get

$$\mu^* - k_\text{u} \;\geq\; \frac{(1-\rho)P + 2k_\text{u}\rho}{(1-\rho) + 2n\rho} \cdot n - k_\text{u} \;=\; \frac{(\mu - k_\text{u})(1-\rho)}{(1-\rho) + 2n\rho}.$$

We exploit that $\rho \leq 1/2$, implying $1 - \rho \geq 0$. Hence, if $\mu - k_u \geq 0$ then $\mu^* \geq k_u > k_u - 1/2$ anyway. Assuming $\mu - k_u < 0$, we can lower bound the (negative) last fraction by $(\mu - k_u)/(2n\rho)$. Hence, if we can prove that $k_u - \mu < n\rho$, we obtain $\mu^* > k_u - 1/2$ as desired. We will bound the probability of a large deviation $k_u - \mu$ keeping track of the variance of the random OneMax-value of x^*. Let v be the variance before the pheromone values have been updated with respect to x^* and denote by v^* the variance after the update. If $v \leq (n\rho)^{3/2}$, then a Chernoff-Hoeffding-type bound (see Appendix A.2.3) yields

$$\text{Prob}(k_u - \mu \geq n\rho) \leq e^{-\frac{(n\rho)^2}{2v(1+n\rho/(3v))}} = 2^{-\Omega(\sqrt{n\rho})} = 2^{-\Omega(n^{\epsilon/2})}.$$

However, we cannot show that $v \leq (n\rho)^{3/2}$ is likely for all points of time. Therefore, we will prove $v^* \geq v/(4n\rho)$ for any time step. This will show that v^* is large enough to compensate a large $k_u - \mu$ in the following step, constructing x.

Suppose $v > (n\rho)^{3/2}$. Then $v \geq \sqrt{vn\rho}$, and the above bound yields

$$\text{Prob}(k_u - \mu \geq \sqrt{vn\rho}) \leq e^{-\frac{(\sqrt{vn\rho})^2}{2v+2\sqrt{vn\rho}/3}} \leq e^{-\frac{vn\rho}{2v+2v/3}} = 2^{-\Omega(n^\epsilon)}.$$

Hence, with probability $1 - 2^{-\Omega(n^\epsilon)}$, $(k_u - \mu)/(2n\rho) \leq \sqrt{v/(2n\rho)}$, implying $\mu^* \geq k_u - \sqrt{v/(2n\rho)}$. Due to the assumptions $v > (n\rho)^{3/2}$, $v^* \geq v/(4n\rho)$ and $n\rho = \Omega(n^\epsilon)$, it follows that $v^* = \omega(1)$. Hence, we can apply Lindeberg's generalization of the Central Limit Theorem for the value of x. The probability of producing at least $k_u + 1$ ones on the edges of $\mathcal{U}$ is bounded below by the probability of producing at least $1 + \mu^* + \sqrt{v/(2n\rho)}$ ones on these edges. By the Central Limit Theorem, this has probability $\Omega(1)$ since $\sqrt{v^*} \geq \sqrt{v/(2n\rho)}$.

We still have to show that $v^* \geq v/(4n\rho)$. It is sufficient to show a statement on the success probability for each edge (u, v) of the construction graph. Consider the expression $\tau'_{(u,v)} \geq \frac{(1-\rho)\tau_{(u,v)}}{1-\rho+2n\rho}$. The last fraction is at least $\frac{\tau_{(u,v)}}{4n\rho}$ since $\rho \leq 1/2$.

The edges of $\mathcal{S}$ contribute with probability $\Omega(1)$ at least k_s, and (if no failure of probability $2^{-\Omega(n^{\epsilon/2})}$ occurs) with probability $\Omega(1/n)$, the value of the bits corresponding to edges of $\mathcal{U}$ is at least $k_u + 1$. At most $n - 1$ improvements are needed, and, by Chernoff bounds, cn^2 steps contain at least $n - 1$ improvements with probability $1 - 2^{-\Omega(n)}$ for an appropriate constant c. Since $\rho \leq 1/2$, $\epsilon \leq 1$ must hold. Hence, the sum of all failure probabilities for $O(n^2)$ steps is $2^{-\Omega(n^{\epsilon/2})}$. □

8.4 Conclusions

For the first time, bounds on the runtime of a simple ACO algorithm have been obtained. Choosing a large evaporation factor, it behaves like (1+1) EA_b and all results on this algorithm transfer directly to our ACO algorithm. In addition, we have inspected the effect of the evaporation factor in detail for the function OneMax and figured out the border between a polynomial and an exponential optimization time almost completely. Thereby,

we have developed new techniques for the analysis of randomized search heuristics.

A conference version that contains the results of this chapter has been submitted for publication.

Part IV

Multi-Objective Optimization Problems

Chapter 9

Multi-Objective Minimum Spanning Trees

Especially in the case of multi-objective optimization problems evolutionary algorithms have been successfully applied. In this case the population of a multi-objective evolutionary algorithm (MOEA) is used to compute or approximate the Pareto front. In contrast to their wide applications the rigorous analysis of MOEAs with respect to their runtime behavior has been started only recently. Such analyses are necessary to get a better understanding how these heuristics work and which parts of the Pareto front can be computed fastly. To our knowledge, there are only a few results on the expected runtime of multi-objective evolutionary algorithms. Laumanns, Thiele, Zitzler, Welzl, and Deb (2002) have analyzed two local search algorithms (SEMO and FEMO) for a problem with conflicting objectives. Giel (2003) has investigated a simple MOEA that searches globally (GSEMO).

In this chapter we analyze MOEAs on a NP-hard multi-objective combinatorial optimization problem. Laumanns, Thiele, and Zitzler (2004) have considered a special instance of the multi-objective knapsack problem. An analysis of MOEAs on a combinatorial problem, not only specific instances, is still missing. The investigations presented in this chapter seem to be the first analyses of a MOEA with respect to the expected runtime on an NP-hard combinatorial optimization problem. We consider the multi-objective minimum spanning tree problem. Many successful evolutionary algorithms have been proposed for this problem (see e. g. Zhou and Gen (1999) or Knowles and Corne (2001)). In Chapter 7 we have shown that randomized search heuristics are able to compute minimum spanning trees in expected polynomial time. The analysis is based on the investigation of the expected multiplicative weight decrease (with respect to the difference of the weight of the current graph and the weight of a minimum spanning tree) and serves as a starting point for the analysis on the multi-objective minimum spanning tree problem. We analyze GSEMO (see Section 4.2) until it has produced a population including for each extremal point of the Pareto front a corresponding solution. The extremal points of the Pareto front are of particular interest as we show that they give a 2-approximation of the Pareto front.

After having given a motivation for our work we introduce in Section 9.1 our model of the multi-objective minimum spanning tree problem. In Section 9.2 we show that the extremal points give a 2-approximation of the Pareto front. We analyze GSEMO in Section 9.3 with respect to the expected time until it has produced a population that includes for each extremal point of the Pareto front a corresponding spanning tree and finish with some conclusions.

9.1 The problem

The problem to compute multi-objective minimum spanning trees can be stated as follows. Given an undirected connected graph $G = (V, E)$ on n vertices and m edges and for each edge $e \in E$ a weight vector $w(e) = (w_1(e), \ldots, w_k(e))$, where $w_i(e)$, $1 \leq i \leq k$, is a positive integer, the goal is to find for each objective vector q of the Pareto front F a spanning tree s with $w(s) = q$. In the case of at least two weight functions the problem is NP-hard (see e.g. Ehrgott (2005)). Papadimitriou and Yannakakis (2000) have given an fully polynomial time approximation scheme (FPTAS) to compute an ϵ-approximation of the Pareto front. This algorithm is based on a pseudo-polynomial algorithm for the problem given by Barahona and Pullyblank (1987). In the rest of this chapter we consider the case $k = 2$ and examine which parts of the Pareto front can be computed by simple MOEAs in pseudo-polynomial time.

Again we use the edge-set encoding for our algorithms. The search space equals $S = \{0,1\}^m$ where each position corresponds to one edge. A search point s corresponds to the choice of all edges e_j, $1 \leq j \leq m$, where $s_j = 1$ holds. Let w_i^{max} be the maximum weight of w_i, $w_{max} = \max w_i^{max}$, $w_{min} = \min w_i^{max}$ and $w_{ub} = n^2 \cdot w_{max}$, The fitness of an individual s is described by a vector $f(s) = (f_1(s), \ldots f_k(s))$ with

$$f_i(s) := (c(s) - 1) \cdot w_{ub}^2 + (e(s) - (n-1)) \cdot w_{ub} + \sum_{j|s_j=1} w_i^j$$

where w_i^j is the value of edge e_j with respect to the function w_i, $c(s)$ is the number of connected components in the graph described by s and $e(s)$ is the number of edges in this graph. This is a generalization of the fitness function w used for RLS_b and (1+1) EA_b in Chapter 7 to the multi-objective case. Again the most important issue is to decrease $c(s)$ until we have graphs connecting all vertices. The next issue is to decrease $e(s)$ under the condition that s describes a connected graph. Finally, we look for Pareto optimal spanning trees.

The fitness function f, similar to the fitness function w of Chapter 7, penalizes the number of connected components as well as the extra connections. This is not necessary since breaking a cycle decreases the fitness value. Therefore we are also interested in the

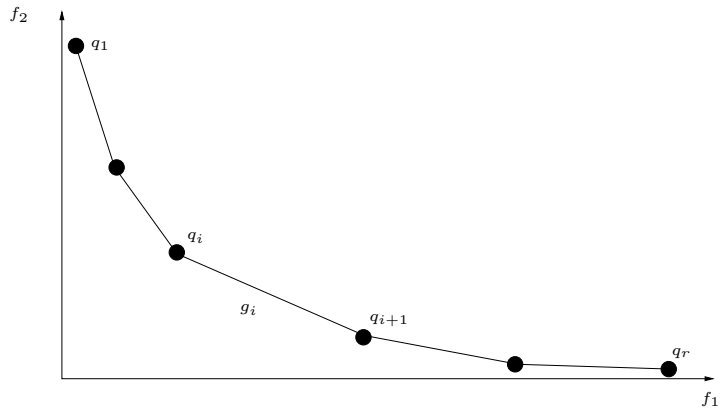

Figure 9.1: The convex hull of the Pareto front F

fitness function $f'(s) = (f'_1(s), \dots f'_k(s))$ with

$$f'_i(s) := (c(s) - 1) \cdot w_{ub} + \sum_{j|s_j=1} w_i^j,$$

which generalizes the fitness function w' of Chapter 7 to the multi-objective case.

Note that the fitness functions f and f' compute the same objective vector if s describes a spanning tree. This implies that the Pareto fronts for a given connected graph G contain the same objective vectors. As both fitness functions take only the weight vectors on the edges into account if s is a spanning tree, the Pareto sets of f and f' consist of all Pareto optimal spanning trees.

Considering a spanning tree T we can create another spanning tree T' by integrating an edge $e \in E \setminus T$ into T and removing one edge of the created cycle $Cyc(T, e)$. Using such local changes we can transform a spanning tree T into another spanning tree S. The properties of such local changes have been examined in detail in Section 7.2. We will also make use of these results to obtain upper bounds on the runtime of simple MOEAs for the multi-objective minimum spanning tree problem.

9.2 The extremal points of the convex hull

Let F be the Pareto front of a given instance. If we consider the bi-objective problem the convex hull of F, denoted by $conv(F)$, is a piecewise linear function (see Figure 9.1). Note that for each spanning tree T on the convex hull there is a $\lambda \in [0, 1]$ such that T is a minimum spanning tree with respect to the single weight function $\lambda w_1 + (1-\lambda)w_2$ (see e.g. Knowles and Corne (2001)). We will use this in Section 9.3 to transform an arbitrary spanning tree S into a desired Pareto optimal spanning tree T on $conv(F)$ using Theorem 7.2.1.

Let q_1 and q_r be the Pareto optimal objective vectors with minimal weight with respect to f_1 respectively f_2. We denote by g_i, $1 \leq i \leq r-1$, the linear functions with gradients m_i describing $conv(F)$. Then $i < j$ holds for two linear functions g_i and g_j iff $m_i < m_j$. Hence, the linear functions are ordered with respect to their increasing gradients. Let $q_i = (x_i, y_i)$, $2 \leq i \leq r-1$, be the intersecting point of g_{i-1} and g_i. Our aim is to analyze the expected time until a simple MOEA has produced a population that includes for each vector of $F' = \{q_1, q_2, \ldots, q_r\}$ a spanning tree. We call the vectors of F' the extremal points of the Pareto front.

The general idea of evolutionary algorithms is to create good approximations of optimal solutions for a given task. In the case of multi-objective problems the task is to approximate the Pareto front.

Definition 9.2.1 *A solution x is called a c-approximation of a solution x^* if $f_1(x) \leq c \cdot f_1(x^*)$ and $f_2(x) \leq c \cdot f_2(x^*)$ holds. We also call the vector $(f_1(x), f_2(x))$ a c-approximation of the vector $(f_1(x^*), f_2(x^*))$ in this case. A set P of solutions is called a c-approximation of the Pareto front F, if there exists for each solution x^* of the Pareto set a solution x in P that is a c-approximation of x^*.*

We are not aware of a result in the literature showing that, in the case of the minimization of two arbitrary functions with positive function values, the extremal points of the Pareto front give a 2-approximation of the Pareto front. Therefore, we show the following result.

Theorem 9.2.2 *Considering the minimization of two objective functions with positive objective values, a set P containing for each extremal point a solution is a 2-approximation of the Pareto front.*

Proof: Consider the different possibilities for an arbitrary solution x^* of the Pareto set together with its Pareto optimal objective vector $q = (q_1, q_2)$, which is not an extremal point. Assume that $x_i < q_1 < x_{i+1}$ holds for some $i \in \{1, \ldots, r-1\}$. This has to be the case because otherwise q would be dominated or the extremal points are not Pareto optimal. The situation for the two extremal points q_i and q_{i+1} is shown in Figure 9.2. If $q_2 \geq y_i$, q is not Pareto optimal as it is dominated by q_i. Let $d_x = x_{i+1} - x_i$ and $d_y = y_{i+1} - y_i$. Consider the vector $s = (s_1, s_2)$ with $s_1 = x_i + (d_x/2)$ and $s_2 = y_i - (d_y/2)$ in the objective space. As a 2-approximation of a vector q that dominates a vector q' is also a 2-approximation of q', it is not necessary to consider the vectors dominated by s. Note, that s is a point on the linear function g_i which separates the possible vectors that have to be considered into two classes class 1 and class 2. Class 1 includes all vectors q with $x_i < q_1 < x_i + d_x/2$ and $y_i - d_y/2 < q_2 < y_i$ and in addition the vector s. Class 2 includes all vectors q with $x_i + d_x/2 < q_1 < x_{i+1}$ and $y_{i+1} < q_2 < y_i - d_y/2$.

Clearly $x_i \leq q_1$ and $y_{i+1} \leq q_2$. In the following we show that if q belongs to class 1, $y_i \leq 2 \cdot q_2$ holds and that if q belongs to class 2, $x_{i+1} \leq 2 \cdot q_1$ holds. Hence, either q_i or q_{i+1} is a 2-approximation of q.

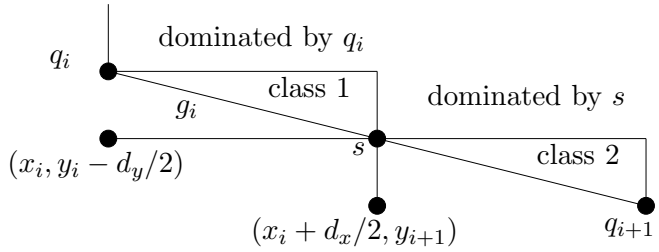

Figure 9.2: Possible Pareto optimal vectors between two extremal points q_i and q_{i+1}

If q belong to class 1 then $q_1 \leq x_i + (d_x/2)$ holds. In this case, we get

$$\begin{aligned} q_2 &\geq y_i + (q_1 - x_i) \cdot m_i \\ &\geq y_i + (d_x/2) \cdot m_i \\ &= y_i + (d_x/2)\frac{y_{i+1} - y_i}{x_{i+1} - x_i} \\ &= y_i + (1/2)(y_{i+1} - y_i) \end{aligned}$$

which implies $2 \cdot q_2 \geq 2y_i + y_{i+1} - y_i \geq y_i$.

If q belongs to class 2 then $q_2 < y_{i+1} - (d_y/2)$ holds. In this case we get

$$\begin{aligned} q_1 &\geq x_{i+1} + (q_2 - y_{i+1}) \cdot (1/m_i) \\ &\geq x_{i+1} - (d_y/2) \cdot (1/m_i) \\ &= x_{i+1} - (d_y/2)\frac{x_{i+1} - x_i}{y_{i+1} - y_i} \\ &= x_{i+1} - (1/2)(x_{i+1} - x_i) \end{aligned}$$

which implies $2 \cdot q_1 \geq 2x_i + x_{i+1} - x_i \geq x_{i+1}$. □

9.3 Analysis of GSEMO

In the definition of GSEMO we have not specified how to choose the first search point. All results in this section hold for an arbitray initial solution. Note, that often the initial solution is chosen uniformly at random from the considered search space.

We start by analyzing GSEMO until it has produced a population consisting of solutions which are connected graphs.

Lemma 9.3.1 *GSEMO working on the fitness function f or f' constructs a population consisting of connected graphs in expected time $O(m \log n)$.*

Proof: Due to the fitness functions no steps increasing the number of connected components are accepted. The current population P consists at each time of solutions having

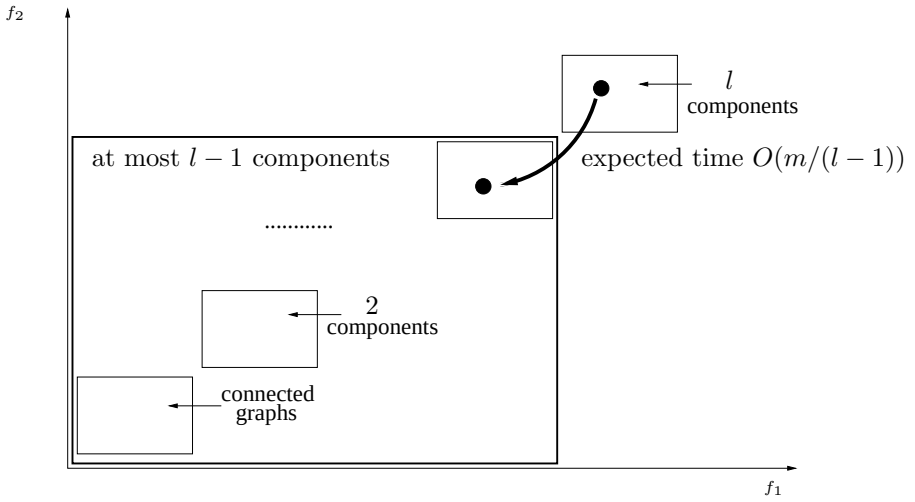

Figure 9.3: Decomposition of the objective space due to the number of connected components

the same number of connected components as otherwise the solution s with the smallest number of connected components would dominate a solution with a larger number of connected components in P. The decomposition of the objective space due to the number of connected components is shown in Figure 9.3. If P consists of search points with l, $l \geq 2$, components there are for each search point in P at least $l-1$ edges whose inclusion decreases the number of connected components. The probability of a step decreasing the number of connected components is therefore at least $\frac{1}{e} \cdot \frac{l-1}{m}$ and its expected waiting time is bounded by $O(m/(l-1))$. After we have decreased the number of connected components for one solution, all solutions with more connected components are deleted from the population. Hence, the expected time until the population consists only of solutions describing connected graphs is upper bounded by

$$em\left(1 + \ldots + \frac{1}{n-1}\right) = O(m \log n). \qquad \square$$

Now we bound the expected time until P includes for the objective vectors q_1 and q_r corresponding solutions. Later these solutions will serve as a basis to collect solutions for the remaining extremal points.

Lemma 9.3.2 *GSEMO working on the fitness function f constructs a population, which includes for each of the objective vectors q_1 and q_r a spanning tree, in expected time $O(m^2 n w_{min}(\log n + \log w_{max}))$.*

Proof: Using Lemma 9.3.1, we work under the assumption that P consists of individuals describing connected graphs. In this case, all individuals of P have the same number of edges. If there are N edges in each solution there are $N-(n-1)$ edges whose exclusion

decreases the number of edges without increasing the number of connected components. Hence, the probability to decrease the number of edges in the next step is at least $\frac{1}{e} \cdot \frac{N-(n-1)}{m}$ and we can bound the expected time to create a population consisting of spanning trees by

$$em\left(1 + \ldots + \frac{1}{m-(n-1)}\right) = O(m \log(m-n+1) = O(m \log n).$$

If P consists of spanning trees the population size is bounded by $(n-1)w_{min}$ because there is only one spanning tree for each value of one single function in the population. We show an upper bound on the expected time to create a population including a spanning tree with vector q_1. The expected optimization time of (1+1) EA_b in the case of one cost function is bounded by $O(m^2(\log n + w_{max}))$; see Theorem 7.3.8. We are working with a population of size $O(nw_{min})$ and consider in each step the individual with the smallest weight with respect to the function w_1. In each step this individual is chosen with probability $\Omega\left(\frac{1}{nw_{min}}\right)$. Following the ideas in the proof of Theorem 7.3.8, we can upper bound the expected time until P includes a spanning tree having minimal weight with respect to w_1 by $O(m^2 n w_{min}(\log n + \log w_1^{max}))$.

It remains to bound the expected time to create from a population with a minimal spanning tree S with respect to w_1, a population with a spanning tree T which is minimal with respect to w_1 and also Pareto optimal. If $|S \setminus T| = k$ holds we can consider pairs of edges s_i, t_i with $s_i \in S \setminus T$ and $t_i \in T \setminus S$ due the bijection given by Theorem 7.2.1. As S and T are both minimum spanning trees with respect to w_1, $w_1(s_i) = w_1(t_i)$ holds for $i = 1, \ldots k$, because otherwise we are able to improve T or S with respect to w_1. This contradicts the assumption that S and T are both minimum spanning trees with respect to w_1. $w_2(t_i) \leq w_2(s_i)$ holds for $i = 1, \ldots, k$, because otherwise we are able to improve T with respect to w_2 without changing the value of w_1. A contradiction to the assumption that T is Pareto optimal. Hence, there are k exchange operations which turn S into T and the expected time to create T from S is bounded by $O(m^2 n w_{min}(\log n + \log w_2^{max}))$ using the ideas in the proof of Theorem 7.3.8.

Altogether we obtain an upper bound of $O(m^2 n w_{min}(\log n + \log w_1^{max} + \log w_2^{max})) = O(m^2 n w_{min}(\log n + \log w_{max}))$ to contruct a spanning with vector q_1. After we have constructed a population including a spanning tree for q_1, we can upper bound the expected time to create a population including for each of the vectors q_1 and q_r a spanning tree by $O(m^2 n w_{min}(\log n + \log w_{max}))$ using the same arguments as before and this proves the lemma. □

We give a similar bound for the fitness function f'. The main difference is that we can only guarantee a population size which is bounded by $O(mw_{min})$.

Lemma 9.3.3 *GSEMO working on the fitness function f' constructs a population, which includes for each of the objective vectors q_1 and q_r a spanning tree, in expected time $O(m^3 w_{min}(\log n + \log w_{max}))$.*

Proof: We consider the expected time to create a spanning tree with vector q_1. At each time the population size is bounded by mw_{min}, because there is only one search point for each value of one single function in the population. We consider in each step the connected graph with the minimal weight with respect to w_1 in P. Using the ideas of Lemma 9.3.2 a connected subgraph with minimal costs with respect to w_1 is constructed in expected time $O(m^3 w_{min}(\log n + \log w_1^{max}))$. This is a spanning tree, because otherwise the weight of w_1 can be decreased. After that we consider the spanning tree with minimal weight with respect to w_1 in P. We are in the situation to minimize the weight of this spanning tree with respect to w_2 and this can be done in expected time $O(m^3 w_{min}(\log n + \log w_2^{max}))$ using the ideas of Lemma 9.3.2. The expected time to create a spanning tree with vector q_r can be bounded in the same way. □

In the following we work under the assumption that $F = F'$ holds which means that the Pareto front consist only of extremal points. In this case we call the Pareto front strongly convex. Let $d(T, T') = |T \setminus T'|$ denote the distance of two spanning trees T and T' which equals the minimal number of exchanges of two edges to construct T' from T.

Lemma 9.3.4 *Assume that the Pareto front F is strongly convex. For each spanning tree T with $w(T) = q_i$, $1 \leq i \leq r-1$, there is a spanning tree T' with $w(T') = q_{i+1}$ and $d(T, T') = 1$.*

Proof: As T and T' are different $d(T, T') > 0$ holds. We assume that T' is a spanning tree with vector q_{i+1} which has minimal distance to T. Working under the assumption that $d(T, T') > 1$ holds for all spanning trees T' with vector q_{i+1}, we show a contradiction. We can apply Theorem 7.2.1 because for each spanning tree T' of the convex hull $conv(F)$ there is a $\lambda \in [0, 1]$ such that T' is a minimum spanning tree for the single weight function $\lambda w_1 + (1-\lambda)w_2$. We partition the different exchange operations $exchange(e, e')$ inserting e and deleting e' due to Theorem 7.2.1 into 4 groups (see Figure 9.4). Let $d = exchange(e, e')$ and $w(d) = (w_1(e) - w_1(e'), w_2(e) - w_2(e'))$ be the vector describing the weight changes of this operation. d belongs to group 1, if $w_1(d) < 0$ and $w_2(d) > 0$, to group 2, if $w_1(d) \geq 0$ and $w_2(d) \geq 0$, to group 3, if $w_1(d) < 0$ and $w_2(d) < 0$, and to group 4, if $w_1(d) > 0$ and $w_2(d) < 0$.

There is no exchange operation d with $w(d) = (0, 0)$, because otherwise T' is not a spanning tree with vector q_{i+1} and minimal distance to T. All other operations belonging to group 2 are not possible because the remaining operations applied to T would construct a spanning tree dominating T'. A contradiction to the assumption that T' is Pareto optimal. Operations belonging to group 3 are not possible because they would construct a spanning tree dominating T. Let $q_i = (x_i, y_i)$, $1 \leq i \leq r$. There is no exchange operation belonging to group 4 which constructs a spanning tree T'' with value $x_i < w_1(T'') < x_{i+1}$, because q_{i+1} lexicographically follows q_i in the Pareto front. There is also no operation belonging to group 4 constructing a spanning tree with value $w_1(T'') \geq x_{i+1}$ and $w_2(T'') \geq y_{i+1}$, because otherwise the remaining operations applied to T construct a spanning tree which dominates T'. A contradiction to the assumption that T' is Pareto optimal.

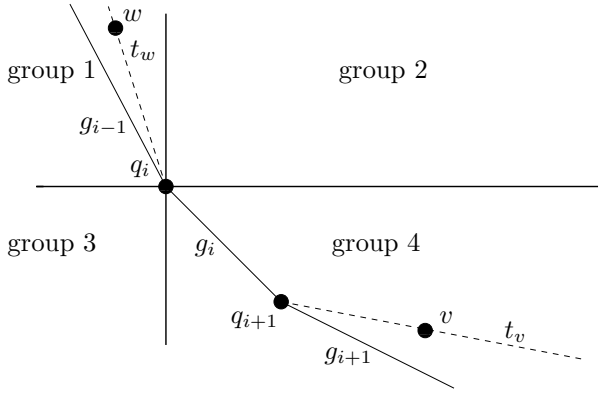

Figure 9.4: The strongly convex Pareto front and the classification of exchange operations creating a spanning tree T' with vector q_{i+1} from a spanning tree T with vector q_i.

Let M be the set of exchange operations constructing T' from T, $M_1 \subseteq M$ be the set of operations belonging to group 4, and $M_2 \subset M$ be the subset of operations belonging to group 1. Note that $M_1 \cup M_2 = M$ holds due to previous observations. We assume that M consists of more than one operation. As $x_{i+1} > x_i$ holds, M_1 is not empty. Let $v = (v_x, v_y)$ be the vector of the spanning tree which is constructed when all operations of M_1 are applied to T. $v_x > x_{i+1}$ and $v_y < y_{i+1}$ holds, because otherwise we have produced a spanning tree with vector q_{i+1} by one single operation (a contradiction to $d(T, T') > 1$), have constructed a spanning tree dominating T', or the remaining operations applied to T construct a spanning tree dominating T'. We consider the linear function t_v with the gradient m_v intersecting the points q_{i+1} and v. As F is strongly convex $m_v \geq m_i > m_{i-1}$ holds. To construct a spanning tree with vector q_{i+1} M_2 can not be empty. Let $w = (w_x, w_y)$ be the vector of the spanning tree which is constructed when the operations of M_2 are applied to T and let t_w be the linear function with gradient m_w defined by q_i and w. As F is strongly convex $m_w \leq m_{i-1}$ holds which implies $m_v > m_w$. Let $z = (z_x, z_y)$, $z_x < 0, z_y > 0$, be the vector such that $q_i + v + z = q_{i+1}$. As the operations of M applied to T construct T' with vector q_{i+1}, $w_x = z_x$ must hold. Taking the gradient m_w into account, we can compute the value of the second component by $v_y + m_w \cdot z_x > v_y + m_v \cdot z_x = y_{i+1}$. A contradiction to the assumption that the operations of M applied to T construct a spanning tree T' with vector q_{i+1}. Hence, T' has to be constructed from T by one single operation belonging to group 4. □

Let $|F|$ be the number of Pareto optimal objective vectors. Note that $|F| \leq (n-1)w_{min}$ holds. In the following, we show an upper bound on the expected time until the population P includes for each vector of a strongly convex Pareto front F a corresponding spanning tree.

Theorem 9.3.5 *The expected time until GSEMO working on the fitness function f or f' has constructed a population, which includes a spanning tree for each vector of a strongly*

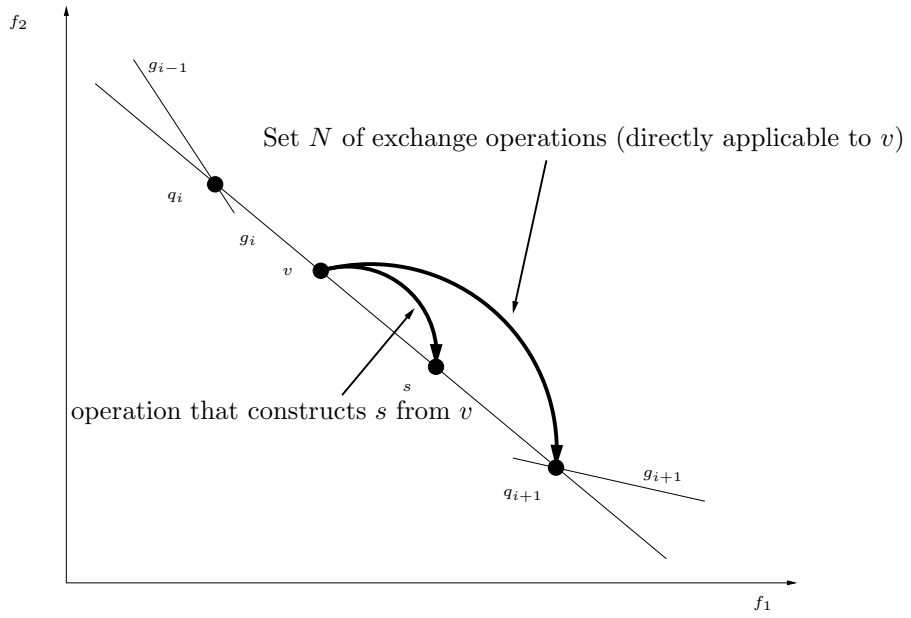

Figure 9.5: Situation to compute the next extremal point

convex Pareto front F, is bounded by $O(m^2nw_{min}(|F| + \log n + \log w_{max}))$ respectively $O(m^3w_{min}(|F| + \log n + \log w_{max}))$.

Proof: We consider the fitness function f. Due to Lemma 9.3.2 the expected time to create a population including spanning trees for the Pareto optimal vectors q_1 and q_r is bounded by $O(m^2nw_{min}(\log n + \log w_{max}))$. We assume that the population includes a spanning tree for each q_j, $1 \leq j \leq i$. For each spanning tree T with vector q_i there exists a spanning tree T' with vector q_{i+1} and $d(T, T') = 1$. The probability to choose the individual representing T in the next mutation step is at least $\frac{1}{(n-1)w_{min}}$, because the population size is bounded by $(n-1)w_{min}$. As $d(T, T') = 1$ holds for at least one spanning tree T' with vector q_{i+1}, the probability to construct such a T', after having chosen the individual x describing T, is at least $\frac{1}{m^2}\left(1 - \frac{1}{m}\right)^{m-2} \geq \frac{1}{em^2}$. Hence, the expected time to create a spanning tree with vector q_{i+1} is bounded by $O(m^2nw_{min})$. As there are $|F|$ Pareto optimal vectors the expected time until GSEMO constructs a spanning tree for each Pareto optimal vector of a strongly convex Pareto front is bounded by $O(m^2nw_{min}(|F| + \log n + \log w_{max}))$. The ideas can be easily adapted to f' using Lemma 9.3.3 and the upper bound mw_{min} on the population size. □

We consider the general case now and give an upper bound on the expected time until GSEMO has constructed a population including a spanning tree for each extremal point $q \in F'$ of an arbitrary Pareto front F. Let $C = conv(F)$ be the set of objective vectors on the convex hull of F. Note that $|C| \leq (n-1) \cdot w_{max}$ holds.

Theorem 9.3.6 *The expected time until GSEMO working on the fitness function f or f' has constructed a population, which includes a spanning tree for each vector $q \in F'$,*

is bounded by $O(m^2nw_{min}(|C| + \log n + \log w_{max}))$ *respectively* $O(m^3w_{min}(|C| + \log n + \log w_{max}))$.

Proof: Again we consider the fitness function f and adapt the ideas to achieve the upper bound for f'. By Lemma 9.3.2 the population P includes after an expected number of $O(m^2nw_{min}(\log n + \log w_{max}))$ steps spanning trees for the vectors q_1 and q_r. To transform a spanning tree of $conv(F)$ into another spanning tree of $conv(F)$ we use the set of exchange operations described by Theorem 7.2.1. Let T be a spanning tree with vector q_i, $1 \leq i \leq r-2$ and suppose that T' is a spanning tree with vector q_{i+1} and minimal distance to T. We denote by M the set of exchanges operations classified as in the proof of Lemma 9.3.4 that construct T' from T. Using the arguments in the proof of Lemma 9.3.4 there are no exchanges belonging to group 2 or 3 in M. We show that each subset of M applied to T constructs a spanning tree on g_i. Suppose that a subset $M' \subseteq M$ of the operations constructs a spanning with a vector v not lying on g_i. This vector has to lie above g_i because otherwise it is outside of $conv(F)$. To construct a spanning tree with vector q_{i+1} on g_i the operations of $M'' = M \setminus M'$ have to construct a spanning lying below g_i. A contradiction to the assumption that g_i is part of $conv(F)$.

We consider the spanning tree T'' with the lexicographic greatest vector $v = (v_x, v_y)$ on g_i in the population (see Figure 9.5). If $v \neq q_{i+1}$ T' can be constructed from T'' by a set N of exchanges of two edges, where each single exchange operation executed on T'' yields a spanning tree with vector on g_i. As $v_x < x_{i+1}$ holds there is at least one operation in this set N which constructs a spanning tree on g_i with vector $s = (s_x, s_y)$ where $v_x < s_x \leq x_{i+1}$ holds. Such a spanning tree is a spanning tree of $conv(F)$. Let C_i be the set of Pareto optimal vectors on g_i, $1 \leq i \leq r-2$, excluding the lexicographic smallest vector and including the lexicographic greatest vector. The expected time to construct from a population P having spanning trees for the vectors of $\{q_1, \ldots, q_i, q_r\}$, $1 \leq i \leq r-2$, a population including spanning trees for the vectors of $\{q_1, \ldots q_i, q_{i+1}, q_r\}$ is therefore upper bounded by $O(m^2nw_{min}|C_i|)$.

As $|C| = 1 + \sum_{i=1}^{r-2} |C_i|$ holds, the expected time, starting with a population including spanning trees for q_1 and q_r, to construct a population including a spanning tree for each vector of F' is bounded by $O(m^2nw_{min}|C|)$. Together with Lemma 9.3.2 we obtain the proposed bound.

To prove the upper bound for f' we use Lemma 9.3.3 and the upper bound of mw_{min} on the population size. Together with previous ideas we obtain an upper bound of $O(m^3w_{min}|C|)$ after having constructed a population which includes spanning trees for q_1 and q_r and this proves the theorem. □

9.4 Conclusions

The multi-objective minimum spanning tree problem is one of the best-known multi-objective combinatorial optimization problems. For the first time evolutionary algorithms have been analyzed with respect to the expected time until they produce solutions of the

Pareto front. In the case of a strongly convex Pareto front, we have achieved a pseudo-polynomial bound on the expected time until the population includes for each Pareto optimal objective vector a corresponding spanning tree. For an arbitrary Pareto front we have considered the extremal points of the Pareto front. These points are of particular interest as they give a 2-approximation of the Pareto front. It has been shown that the population includes a solution for each extremal point after a pseudo-polynomial number of steps.

A conference version has been published in the Proceedings of Parallel Problem Solving from Nature (PPSN) 2004 (see Neumann (2004b)). An extended journal version that contains the results of this chapter will appear in a Special Issue on Evolutionary Multi-Objective Optimization of the European Journal on Operational Research (see Neumann (2007)).

Chapter 10

Minimum Spanning Trees Made Easier

In the previous chapter, we have analyzed simple MOEAs on a given multi-objective optimization problem. In this chapter, we consider a multi-objective model of the minimum spanning tree problem. A single-objective model for the computation of minimum spanning trees has already been examined in Chapter 7. Our goal is to show that sometimes single-objective optimization problems can be solved much easier by using a multi-objective model of the problem. This approach may open a new research area in the field of multi-objective optimization.

Sometimes, people try to turn multi-objective problems into single-objective ones, e. g., by optimizing a weighted sum of the fitness values of the single criteria. This may be useful in some applications but, in general, we do not obtain the information contained in the Pareto front and corresponding search points. Many variants of evolutionary algorithms specialized to multi-objective optimization problems have been developed and applied, for a survey see the monographs of Deb (2001) and Coello Coello, Van Veldhuizen, and Lamont (2002). A conclusion from this discussion is that "multi-objective optimization is more (at least as) difficult than (as) single-objective optimization". This is true at least if the fitness values for the different criteria are "somehow independent". Without such an assumption there is no reason to believe in the conclusion above.

We discuss the following scenario. The considered problem is a single-objective problem. It is possible to add some further criteria such that the Pareto front of the newly created multi-objective optimization problem is not too large and such that the solution of the multi-objective problem includes the solution of the single-objective problem. Solving the multi-objective problem instead of the single-objective problem implies to compute the Pareto front instead of a single optimal value. Each considered search point contains more information than in the single-objective case since it contains also the fitness values for the additional criteria. At least in principle it is possible that this additional information improves the search behavior of evolutionary algorithms. This would imply that for solving

difficult single-objective optimization problems one should also think about the possibility to model the problems as generalized multi-objective optimization problems.

The purpose of this chapter is to prove that the considered scenario is not a fiction. We do not investigate artificial problems to support this claim but one of the combinatorial optimization problems contained in each textbook namely the computation of minimum spanning trees. (Nobody should expect that evolutionary algorithms computing minimum spanning trees beat the well-known problem-specific algorithms.) In Chapter 7, we have already considered the runtime behavior of RLS_b and (1+1) EA_b on this problem.

In Section 10.1, we introduce the two-objective variant of the minimum spanning tree problem which is subject of our investigations and distinguish it from other multi-objective variants of the minimum spanning tree problem. In Section 10.2, we prove upper bounds on the expected optimization time of some evolutionary algorithms for multi-objective optimization applied to our problems. It turns out that they are asymptotically smaller than lower bounds for the worst-case instances of simple evolutionary algorithms for the single-objective case. In order to investigate what happens for small problem dimensions and typical problem instances we have performed several experiments whose results are presented in Section 10.3. We finish with some conclusions.

10.1 A two-objective model

In Chapter 7 we have considered RLS_b and (1+1) EA_b for the minimum spanning tree problem. We have penalized edge sets which do not describe connected graphs (and in one model additionally edge sets containing cycles) and have proven the following results:

- The expected optimization time of RLS_b and (1+1) EA_b is $O(m^2(\log n + \log w_{max}))$ where w_{max} is the largest weight of the considered graph.
- There are graphs with n vertices, $m = \Theta(n^2)$ edges, and $w_{max} = \Theta(n^2)$ such that the expected optimization time of RLS_b and (1+1) EA_b equals $\Theta(m^2 \log n)$.

We discuss the reason for the expected optimization time of RLS_b and (1+1) EA_b. If a search point describes a non-minimum spanning tree, one-bit flips are not accepted. Either the new search point describes an unconnected graph or a connected graph with a larger weight. We have to wait until a mutation step includes an edge and excludes a heavier one from the newly created cycle. The expected waiting time for a specified 2-bit flip equals $\Theta(m^2)$.

As already mentioned, the considered algorithms penalize the number of connected components. This motivates the following two-objective optimization model of the minimum spanning tree problem.

- The search space S equals $\{0,1\}^m$ for graphs on m edges and the search point s describes an edge set.
- The fitness function $f : S \to \mathbb{R}^2$ is defined by $f(s) = (c(s), w(s))$ where $c(s)$ is the number of connected components of the graph described by s and $w(s)$ is the total weight of all chosen edges.
- Both objectives have to be minimized.

We state some simple properties of this problem that are direct consequences of the presented model.

- The parameter $c(s)$ is an integer from $\{1, \ldots, n\}$.
- The first property implies that the populations of SEMO and GSEMO contain at most n search points and the Pareto front contains exactly n elements.
- The parameter $w(s)$ is an integer.

We have to be careful when discussing this model of the minimum spanning tree problem. In Chapter 9, we have discussed another type of multi-objective minimum spanning tree problem. Each edge has k different types of weights, i. e., $w(e) = (w_1(e), \ldots, w_k(e))$. Unconnected graphs are penalized and the aim is to minimize $f(s)$ where s is not legal if s does not describe a connected graph and $f(s)$ is the sum of all $w(e_i)$ where $s_i = 1$, otherwise. Similarly to other optimization problems this multi-objective variant of a polynomially solvable problem is NP-hard (Ehrgott (2000)). This problem has been attacked in different ways, e. g., by Hamacher and Ruhe (1994). Zhou and Gen (1999) present experimental results for evolutionary algorithms. In Chapter 9 we have analyzed which parts of the Pareto front can be obtained in expected pseudo-polynomial time by GSEMO.

10.2 The analysis of the expected optimization time

The results we will prove hold for SEMO as well as for GSEMO. The essential steps are 1-bit flips. In the definition of SEMO and GSEMO we have not specified how to choose the first search point. We discuss two possibilities.

- The first search point is chosen uniformly at random. This is the typical choice for evolutionary algorithms.
- The first search point is $s = 0^m$ describing the empty edge set. This is quite typical, e. g., for simulated annealing.

Our analysis is simplified by knowing that P contains 0^m. Note that $f(0^m) = (n, 0)$ belongs to the Pareto front and 0^m is the only search point s with $c(s) = n$. First, we

investigate the expected time until the population contains the empty edge set.

One might expect that we only have to wait until all edges of the initial search point s have been excluded. This is not true. It is possible that we accept the inclusion of edges since this decreases the number of connected components (although it increases the total weight). Later, we may exclude edges of the new search point s' without increasing the number of connected components. It is possible to construct a search point s'' which dominates s. Then s is eliminated and all search points in the population (perhaps only one) have more edges than s.

Hence, the situation is more complicated. Instead of the minimal number of edges of all search points in P we analyze the minimal weight of all search points in P. One search point s^* with minimal weight has the largest number of connected components (otherwise, the search point s^{**} with $c(s^{**}) > c(s^*)$ is dominated by s^* and will be excluded from P). We analyze $w(s^*)$. We have reached the aim of our investigations if $w(s^*) = 0$, since this implies $s^* = 0^m$. After initialization, $w(s^*) \leq W := w_1 + \cdots + w_m \leq m \cdot w_{max}$.

Theorem 10.2.1 *Starting with an arbitrary search point the expected time until the population of SEMO or GSEMO contains the empty edge set is* $O(mn(\log n + \log w_{max}))$.

Proof: We only investigate steps where the solution with minimal weight s^* is chosen for mutation. The probability of such a step is always at least $1/n$, since $|P| \leq n$. Hence, the expected time is only by a factor of at most n larger than the expected number of steps where s^* is chosen.

By renumbering, we may assume that s^* has chosen the first k edges. We investigate only steps flipping exactly one bit. This has probability 1 for SEMO and probability at least e^{-1} for GSEMO. These steps are accepted if they flip one of the first k edges. If the edge i is flipped, we obtain a search point whose weight is $w(s^*) - w_i$ and the minimal weight has been decreased by a factor of $1 - \frac{w_i}{w(s^*)}$. The average factor of the weight decrease equals

$$\frac{1}{m}\left(\sum_{1 \leq i \leq k}(1 - \frac{w_i}{w(s^*)}) + \sum_{k+1 \leq i \leq m} 1\right) = 1 - \frac{1}{m}$$

if the choice of a non-existing edge is considered as a weight decrease by a factor of 1. The result $1 - \frac{1}{m}$ does not depend on the population. After $M := \lceil (\ln 2) \cdot m \cdot (\log W + 1) \rceil$ steps choosing the current s^*, the expected weight of the new s^* is bounded above by $(1 - 1/m)^M \cdot W \leq \frac{1}{2}$. Applying Markov's inequality, the probability that $w(s^*) \geq 1$ is bounded above by $1/2$. Hence, $w(s^*) < 1$ holds with probability at least $1/2$. Since weights are integers, $w(s^*) < 1$ implies $w(s^*) = 0$. The expected number of phases of length M until $w(s^*) = 0$ is at most 2. Hence, altogether the expected waiting time for $s^* = 0^m$ is bounded above by $2 \cdot n \cdot M = O(mn(\log n + \log w_{max}))$ for SEMO. The corresponding value for GSEMO is larger at most by a factor of 3. □

One may expect that the upper bound given in Theorem 10.2.1 is not exact for many graphs and starting points.

After having analyzed the expected time to produce a population that includes the empty edge set, we analyze to expected optimization under the condition that the empty edge set is included in the population.

Theorem 10.2.2 *Starting with a population containing the empty edge set the expected optimization time of SEMO or GSEMO is* $O(mn^2)$.

Proof: As long as the algorithm has not reached its goal we consider the smallest i such that the population contains for each j, $i \leq j \leq n$, a Pareto optimal search point s_j with $f(s_j) = (j, w(s_j))$. This implies that the graph described by s_j consists of j connected components and has the minimal possible weight among all possible search points describing graphs with j connected components. After initialization, the population includes 0^m which has the smallest weight among all search points representing graphs with n connected components. Hence, i is well defined. The search point s_j is only excluded from the population if a search point s'_j with $f(s'_j) = f(s_j)$ is included in the population. Hence, the crucial parameter i can only decrease and the search is successful if $i = 1$.

Finally, we investigate the probability of decreasing i. It is well-known that a solution with $i-1$ components and minimal weight can be constructed from a solution with i components and minimal weight by introducing the lightest edge that does not create a cycle. Therefore, it is sufficient to choose s_i for mutation (probability at least $1/n$) and to flip exactly one bit concerning a lightest edge connecting two components in the graph described by s_i (probability at least $1/m$ for SEMO and at least $1/(em)$ for GSEMO). Hence, the expected waiting time to decrease the parameter i is bounded above by $O(nm)$. After at most $n-1$ of such events the search is successful. □

Corollary 10.2.3 *If the weights are bounded above by* 2^n*, SEMO and GSEMO find the Pareto front in the two-objective variant of the minimum spanning tree problem in an expected number of* $O(mn^2)$ *rounds independently from the choice of the first search point.*

For dense graphs, this bound beats the bound $O(m^2 \cdot \log n)$ for the application of RLS_b and (1+1) EA_b to the single-objective variant of the minimum spanning tree problem.

10.3 Experimental results

The theoretical results are asymptotic ones. They reveal differences for worst-case instances and large m. We add experimental results that show what happens for typical instances and reasonable m. In order to compare randomized algorithms on perhaps randomly chosen instances one may compare the average run times, but these values can be highly influenced by outliners. We have no hypothesis about the probability distribution describing the random run time for constant input length. Hence, only parameter-free statistical tests can be applied. We apply the Mann-Whitney test (MWT) (see e. g. Swinscow

and Campbell (2001)) that ranks all observed run times. Small ranks correspond to small run times. If the average rank of the results of algorithm A_1 are smaller than those for A_2, MWT decides how likely it can be that such a difference or a larger one can occur under the assumption that A_1 is not more efficient than A_2. If the corresponding p-value is at most 0.05, we call the result significant, for 0.01 very significant, and for 0.001 highly significant. The statistical evaluation has been performed with the software SPSS (Version 11.5, see www.spss.com). The tables contain the considered class of graphs, the average rank AR of different algorithms and the p-value for the hypothesis that the algorithm with the smaller AR-value is likely to be faster.

The experiments consider the following graph classes.

- uniform$_n$: these are complete graphs with $m = \binom{n}{2}$ edges and the weights are chosen independently and uniformly at random from $\{1, \ldots, n\}$.

- uniformbd$_n$: each possible edge is chosen with probability $3/n$ leading to a small average degree of 3, unconnected graphs are rejected and the construction is repeated, the weights of existing edges are chosen as for uniform$_n$.

- plane$_n$: the n vertices are placed randomly on the points of the two-dimensional grid $\{1, \ldots, n\} \times \{1, \ldots, n\}$, the weight of an edge is the rounded Euclidean distance between the vertices.

- planebd$_n$: the n vertices are placed as for plane$_n$ but each edge is only considered with probability $3/n$ as for uniformbd$_n$.

These graph classes reflect different choices of weights (one non-metric and one metric one) and the possibility of dense and sparse graphs. Our algorithms are RLS$_b$, (1+1) EA$_b$, SEMO, and GSEMO. The index z denotes the case that the initial search point is the empty edge set (or all-zero string). Without an index the initial search point is chosen uniformly at random. The run time of RLS$_b$ and (1+1) EA$_b$ denotes the number of fitness evaluations until a minimum spanning tree is constructed. The run time of SEMO and GSEMO denotes the number of rounds until, in one experiment, P contains a minimum spanning tree or until $f(P)$ equals the Pareto front. In each experiment the compared algorithms are considered for 100 runs leading to an average rank of 100.5.

We have analyzed the influence of the initial search point. First, we have considered the time until the Pareto front is computed. The results are shown in Table 10.1 and can be summarized as follows.

Result 10.3.1 *In* 23 *out of* 24 *experiments the variant starting with the empty edge set has the smaller AR-value. Only* 8 *results are significant, among them* 5 *very significant and* 2 *of these highly significant.*

Table 10.1: Comparison of SEMO and GSEMO with different initial solutions until they have computed the Pareto front

Class	AR SEMO$_z$	AR SEMO	p-value	AR GSEMO$_z$	AR GSEMO	p-value
$uniform_{12}$	92.76	108.25	0.058	89.35	111.66	**0.006**
$uniform_{16}$	83.51	117.49	$<$ **0.001**	91.28	109.72	**0.024**
$uniform_{20}$	99.12	101.89	0.735	94.21	106.80	0.124
$uniform_{24}$	98.01	102.99	0.543	93.65	107.35	0.094
$uniform_{28}$	94.62	106.38	0.151	94.48	106.52	0.141
$uniform_{32}$	91.24	109.76	**0.024**	96.76	104.24	0.361
$plane_{12}$	81.61	119.39	$<$ **0.001**	88.14	112.86	**0.003**
$plane_{16}$	94.51	106.49	0.143	89.38	111.63	**0.007**
$plane_{20}$	97.17	103.83	0.416	95.15	105.85	0.191
$plane_{24}$	93.33	107.67	0.080	103.11	97.89	0.524
$plane_{28}$	90.58	110.43	**0.015**	93.09	107.91	0.070
$plane_{32}$	94.55	106.45	0.146	97.44	103.56	0.455

If we are only interested in the computation of a minimum spanning tree, one may expect that one sometimes computes a minimum spanning tree without computing the empty edge set. Indeed, the influence of the choice of the initial search point gets smaller. For the classes uniform$_n$, $n = 4i$ and $3 \leq i \leq 11$, there is no real difference between SEMO$_z$ and SEMO, while the AR-values of GSEMO are in 8 of the 9 experiments smaller than for GSEMO$_z$. For the classes plane$_n$, $n = 4i$ and $3 \leq i \leq 11$, SEMO$_z$ beats SEMO (7 cases) and GSEMO$_z$ beats GSEMO (7 cases). We do not show the results in detail since they are not significant (with the exception of 3 out of 36 cases). The remaining experiments consider the more general case of an initial search point chosen uniformly at random.

We have not considered the worst-case instances for RLS$_b$ and (1+1) EA$_b$ presented in Chapter 7. This would be unfair against these algorithms. Nevertheless, the experiments of Briest et al. (2004) have indicated that, for n and m of reasonable size, dense random graphs are even harder than the asymptotic worst-case examples. This leads to the conjecture that SEMO beats RLS$_b$ and GSEMO beats its counterpart (1+1) EA$_b$. Here, the run time measures the rounds until a minimum spanning tree is constructed. Table 10.2 proves that our conjecture holds for the considered cases. Note that the average rank of 100 runs of one algorithm is at least 50.5. In several experiments the AR-value of SEMO or GSEMO comes close to this value. For $n \geq 20$ all values are at most 51.6 and for small values of n the AR-values are smaller than 60. We can state the following result.

Result 10.3.2 *It is highly significant for all considered graph classes and graph sizes that SEMO outperforms RLS$_b$ and GSEMO outperforms (1+1) EA$_b$.*

The theoretical analysis of the algorithms gives values of $O(m^2 \log n)$ for RLS$_b$ and (1+1) EA$_b$ and $O(mn^2)$ for SEMO and GSEMO (if the weights are reasonably bounded).

Table 10.2: Comparison of SEMO and GSEMO with their single-criteria counterparts on complete uniform and complete geometric instances

Class	AR RLS_b	AR SEMO	p-value	AR (1+1) EA_b	AR GSEMO	p-value
$uniform_{12}$	146.36	54.64	$< \mathbf{0.001}$	147.79	53.32	$< \mathbf{0.001}$
$uniform_{16}$	148.45	52.55	$< \mathbf{0.001}$	149.28	51.72	$< \mathbf{0.001}$
$uniform_{20}$	149.74	51.26	$< \mathbf{0.001}$	149.40	51.60	$< \mathbf{0.001}$
$uniform_{24}$	150.00	51.00	$< \mathbf{0.001}$	150.29	50.71	$< \mathbf{0.001}$
$uniform_{28}$	150.40	50.60	$< \mathbf{0.001}$	150.23	50.77	$< \mathbf{0.001}$
$uniform_{32}$	150.50	50.50	$< \mathbf{0.001}$	150.50	50.50	$< \mathbf{0.001}$
$plane_{12}$	141.43	59.58	$< \mathbf{0.001}$	145.04	55.96	$< \mathbf{0.001}$
$plane_{16}$	144.25	56.75	$< \mathbf{0.001}$	148.28	52.72	$< \mathbf{0.001}$
$plane_{20}$	149.47	51.53	$< \mathbf{0.001}$	149.54	51.46	$< \mathbf{0.001}$
$plane_{24}$	149.95	51.05	$< \mathbf{0.001}$	149.89	51.11	$< \mathbf{0.001}$
$plane_{28}$	150.40	50.60	$< \mathbf{0.001}$	150.36	50.64	$< \mathbf{0.001}$
$plane_{32}$	150.34	50.66	$< \mathbf{0.001}$	150.28	50.72	$< \mathbf{0.001}$

For complete graphs, $m = \Theta(n^2)$ and we get values $n^4 \log n$ vs. n^4. For sparse graphs, $m = \Theta(n)$ and we get values $n^2 \log n$ vs. n^3. Although these are only upper bounds, one may expect different results for the sparse graphs from uniformbd_n and planebd_n. Table 10.3 shows that this is indeed the case and we obtain the following result.

Result 10.3.3 *It is highly significant for* uniformbd_n *and* $n \geq 24$ *and for* planebd_n *and* $n \geq 16$ *(and the considered values of* n*) that* RLS_b *outperforms SEMO. Similar results hold for (1+1)* EA_b *and GSEMO, but the results are highly significant only for large values of* n*, namely* $n \geq 32$ *for both graph classes.*

Note that the last group of experiments considers values of n up to 100.

10.4 Conclusions

It has been investigated whether the multi-objective variant of a single-variant optimization problem can lead to more efficient optimization processes. This is indeed the case for the well-known minimum spanning tree problem and randomly chosen dense graphs. For sparse connected graphs it is better to use the single-objective variant of the problem. The results are obtained by a rigorous asymptotic analysis of the expected optimization time and by experiments on graphs of reasonable size.

Recently, Gerhards (2006) has investigated a multi-objective model for the degree-constrained minimum spanning tree problem, where the Pareto front may be exponentially large. His experiments do not show any advantage for his multi-objective model. In the

Table 10.3: Comparison of SEMO and GSEMO with their single-criteria counterparts on uniform and geometric instances with bounded average degree

Class	AR RLS_b	AR SEMO	p-value	AR (1+1) EA_b	AR GSEMO	p-value
$uniformbd_{12}$	91.91	109.09	**0.036**	101.44	99.57	0.819
$uniformbd_{16}$	90.62	110.39	**0.016**	103.54	97.46	0.458
$uniformbd_{20}$	89.79	111.22	**0.009**	98.98	102.02	0.710
$uniformbd_{24}$	73.19	127.82	$<$ **0.001**	91.53	109.47	**0.028**
$uniformbd_{28}$	78.01	122.99	$<$ **0.001**	93.03	107.98	0.068
$uniformbd_{32}$	77.92	123.08	$<$ **0.001**	80.85	120.15	$<$ **0.001**
$uniformbd_{40}$	73.02	127.98	$<$ **0.001**	84.37	116.63	$<$ **0.001**
$uniformbd_{60}$	65.40	135.60	$<$ **0.001**	71.22	129.78	$<$ **0.001**
$uniformbd_{80}$	56.70	144.30	$<$ **0.001**	58.72	142.28	$<$ **0.001**
$uniformbd_{100}$	54.99	146.01	$<$ **0.001**	58.47	142.53	$<$ **0.001**
$planebd_{12}$	97.56	103.45	0.472	105.24	95.77	0.247
$planebd_{16}$	81.88	119.13	$<$ **0.001**	96.79	104.22	0.364
$planebd_{20}$	81.06	119.95	$<$ **0.001**	101.70	99.30	0.769
$planebd_{24}$	84.45	116.55	$<$ **0.001**	86.52	114.48	**0.001**
$planebd_{28}$	81.94	119.06	$<$ **0.001**	88.45	112.55	**0.003**
$planebd_{32}$	71.53	129.47	$<$ **0.001**	80.86	120.14	$<$ **0.001**
$planebd_{40}$	67.18	133.82	$<$ **0.001**	74.57	126.44	$<$ **0.001**
$planebd_{60}$	56.59	144.41	$<$ **0.001**	60.69	140.31	$<$ **0.001**
$planebd_{80}$	52.98	148.02	$<$ **0.001**	59.60	141.40	$<$ **0.001**
$planebd_{100}$	52.21	148.79	$<$ **0.001**	52.30	148.70	$<$ **0.001**

case of the vertex cover problem, Glaser (2006) has presented a model where the Pareto front is polynomially bounded in the number of vertices and shown that a multi-objective model can lead to better results on random bipartite graphs. In the next chapter, we will show that a multi-objective view on an NP-hard single-objective optimization problem can also help to come up with faster approximation algorithms.

A conference version that contains the results of this chapter has been published in the Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2005 (see Neumann and Wegener (2005)). A journal version has been accepted for publication in Natural Computing (see Neumann and Wegener (2006)).

Chapter 11

NP-hard Spanning Forest Problems

In the previous chapter, we have shown that randomized search heuristics find a minimum spanning tree much easier in a multi-objective model than in a single-objective one. In this chapter we consider two NP-hard spanning forest problems and use a multi-objective formulation to obtain faster approximation algorithms. Given a undirected connected graph $G = (V, E)$ with n vertices and m edges and positive integer weights $w(e)$ for each edge $e \in E$, we are searching (i) for minimum spanning forests of minimum degree, and (ii) for minimum spanning forests obeying given degree bounds on the vertices. A forest with i connected components is a cycle-free subgraph of G that contains exactly $n - i$ edges. A minimum spanning forest with i connected components is a forest where the sum over all edge weights is minimal among all spanning forests with i connected components. In a minimum spanning forest of minimum degree, the largest vertex degree is as small as possible. This generalizes the problem of computing minimum spanning trees of minimum degree. For our algorithms it is, in contrast to previous work, not necessary to assume the graph to be connected. For simplicity we work under this assumption, but in the case that G is not connected our algorithms would produce solutions for each possible value of i. Note that solutions for the different number of connected components may be of additional interest when each spanning tree has a weight that is not acceptable in practical applications such that the graph has to be partitioned into different clusters. Having a solution for each number of connected components the designer of a network can decide how to build these clusters.

Let Δ^* be the maximum vertex degree of an optimal solution. When edge weights are not considered, or assumed to be uniform, a Δ^*+1 approximation algorithm for minimizing the degree of spanning trees has been obtained by Fürer and Raghavachari (1994). For the weighted case, Fischer (1993) has presented an approximation algorithm that computes a minimum spanning tree of degree at most $b \cdot \Delta^* + \lceil \log_b n \rceil$ in time $O(n^{4+1/\ln b})$ for any $b > 1$, which is the best-known algorithm for this problem up to now. His algorithm is an adaptation of a local search algorithm of Fürer and Raghavachari (1992) to the weighted case. The idea of the local search is to perform edge exchanges until the spanning tree is locally optimal.

We model the problem of computing minimum spanning forests of minimum degree as a multi-objective optimization problem where one objective is to minimize the number of connected components and the other objective to minimize the weight and degree. Our aim is to compute for each i, $1 \leq i \leq n$, a minimum spanning forest with i connected components that has the same approximation quality as the algorithm of Fischer. Our algorithm can be seen as extension of Kruskal's algorithm for the computation of minimum spanning trees and runs in time $O(n^{3+1/\ln b})$. Note that during the run, Kruskal's algorithm computes solutions that are minimum spanning forests for each possible number of connected components. The working principle of our algorithm is also to start with an empty graph and to compute the minimum spanning forests as in the run of Kruskal's algorithm one after another. After a new edge has been introduced that leads to a minimum spanning forest with smaller number of connected components, the degree of this minimum spanning forest is improved by edge exchanges as long as we cannot guarantee our desired approximation quality.

Könemann and Ravi (2003) have considered the problem of approximating minimum spanning trees with nonuniform degree bounds. Given degree bounds B_v for all vertices, they have presented an algorithm that runs in time $O(n^6 \log n)$ to compute a spanning tree in which the degree of each vertex v is $O(B_v + \log n)$ and the weight is by at most a constant factor higher than the weight of any spanning tree that obeys the given degree constraints, if such a tree exists. Note, that the question whether there exists a spanning tree obeying the given degree bounds is already NP-complete. Starting with the empty edge set, we compute in time $O(n^{3+2/\ln b})$ for each number of connected components a minimum spanning forest with the same approximation quality as Könemann and Ravi. For $b > 1$ a large constant the runtime of our algorithm approaches the upper bound $O(n^3)$. An additional advantage of our algorithm is that we do not need the assumption that there is a spanning tree obeying the given degree bounds. If this is not the case our algorithm will produce a set of solutions that contains for each i where a spanning forest exists that respects the degree constraints a solution that has the stated approximation quality.

This chapter is organized as follows. In Section 11.1, we introduce our model for the computation of minimum spanning forests with minimum degree and give a new algorithm for minimizing the degree of minimum spanning forests that runs in time $O(n^{3+1/\ln b})$. Section 11.2 applies our technique in combination with an extension of the primal-dual method for minimum spanning trees by Könemann and Ravi to the problem of computing minimum spanning forests with nonuniform degree bounds. We finish with some conclusions.

11.1 Minimizing the maximum degree

We take a multi-objective view on the computation of minimum spanning forests with minimum degree. Let $S = \{0,1\}^m$ be the search space. A search point $x \in S$ describes

the set of all edges e_i where $x_i = 1$ holds. Let $c(x)$ be the number of connected components of the solution x, $w(x)$ be the weight of the chosen edges, $d_j(x)$ be the number of vertices of degree j in x, and $\Delta(x)$ the maximum vertex degree of x. The objective function value of x is given by the vector $f(x) = (f_1(x), f_2(x))$, where $f_1(x) = c(x)$ and $f_2(x) = (w(x), d_{n-1}(x), \ldots, d_0(x))$. Both objectives f_1 and f_2 have to be minimized. Minimizing the second objective means minimization with respect to the lexicographic order. This generalizes the model we have examined in Chapter 10 for the computation of minimum spanning trees by randomized search heuristics. There we have shown that a multi-objective view leads to a more efficient optimization process than in the case of a single objective one.

Let $f(S)$ be the image of the search space under the objective function f defined above. By intersecting the canonic order on $f_1(S)$ with the lexicographic order on $f_2(S)$, both of which are total orders, a partial order on $f(S)$ can be defined as

$$f(x) \preceq f(x') :\Leftrightarrow f_1(x) \leq f_1(x') \wedge f_2(x) \leq_{\text{lex}} f_2(x') \tag{11.1}$$

for all $x, x' \in S$. This partial order represents our preference relation regarding the solutions. The aim is compute the Pareto front which means to identify all minimal elements of $(f(S), \preceq)$, and with each minimal element one of its pre-images from S.

As the edge weights are positive, a minimum spanning forest with i connected components has a smaller weight than a minimum spanning forest with $i+1$ connected components. Therefore, $(f(S), \preceq)$ has n minimal elements, representing for each i, $1 \leq i \leq n$, a minimum spanning forest with i connected components and minimum degree. Our goal is to approximate the set of minimal elements as good as possible. We want to compute for each i a minimum spanning forest with i connected components that has degree at most $b \cdot \Delta_i^* + \lceil \log_b n \rceil$, where Δ_i^* is the smallest maximum degree of any minimum spanning forest with i connected components.

Fischer's algorithm (1993) for the computation of a minimum spanning tree with degree at most $b \cdot \Delta^* + \lceil \log_b n \rceil$ starts with an arbitrary minimum spanning tree and improves the degree of high-degree vertices. The number of these improvements is bounded by $O(n^{2+1/\ln b})$. A better bound on the number of necessary improvements would yield a better upper bound for the runtime of Fischer's algorithm. We consider the number of necessary improvements for our multi-objective model and start with some general properties of minimum spanning forests with i connected components.

Lemma 11.1.1 *Let Δ_i^*, $1 \leq i \leq n$, be the minimum degree of a minimum spanning forest with i connected components. Then $\Delta_n^* \leq \Delta_{n-1}^* \leq \ldots \leq \Delta_1^*$ holds.*

Proof: Suppose that $\Delta_i^* > \Delta_{i-1}^*$ holds for some $i \in \{2, \ldots n\}$. Let F_{i-1}^* be a minimum spanning forest with $i-1$ connected components and minimum degree. Then we can delete the heaviest edge from F_{i-1}^* to construct a minimum spanning forest with i connected components whose degree is at most Δ_{i-1}^*. This contradicts the assumption. □

Let s_i be a solution with i connected components and minimal weight. We call s_i locally optimal if there is no solution s_i' with $c(s_i') = c(s_i)$ and Hamming distance 2 that is better than s_i with respect to $f_2(s_i)$ when disregarding all $d_j(s_i)$ with $j < \Delta(s_i) - \lceil \log_b n \rceil$. Otherwise, we say that s_i' improves s_i. For the case $i = 1$ Fischer has shown that if there is no improvement for s_1 then the minimum spanning tree has already degree at most $b \cdot \Delta_1^* + \lceil \log_b n \rceil$ for any $b > 1$. We generalize this approximation guarantee of local optimal minimum spanning trees given by Fischer to locally optimal minimum spanning forests.

Lemma 11.1.2 *Let s_i be a solution that is locally optimal and Δ_i be its maximum degree. Then $\Delta_i \leq b \cdot \Delta_i^* + \lceil \log_b n \rceil$ holds for any constant $b > 1$.*

Proof: Consider a locally optimal forest F described by s_i. Let U_i be the set of vertices of degree at least i in F. The number of vertices in U_i is at most n for each i. Hence, the ratio $\frac{|U_{i-1}|}{|U_i|}$ cannot be greater than b for $\log_b n$ consecutive values of i.

Consider a δ with $\Delta_i - \lceil \log_b n \rceil \leq \delta \leq \Delta_i$ such that $\frac{|U_{\delta-1}|}{|U_\delta|} \leq b$. Deleting all edges from F that are adjacent to vertices of U_δ yields a forest F_δ with at least $(\delta - 1)|U_\delta| + 1 + i$ connected components. As F is locally optimal, only edges adjacent to vertices of $U_{\delta-1}$ may participate in a minimum spanning forest with i connected components. Hence, there must be at least $(\delta - 1)|U_\delta| + 1 \geq (\delta - 1)(|U_{\delta-1}|/b) + 1$ edges adjacent to vertices of $U_{\delta-1}$. The average degree of a vertex in $U_{\delta-1}$ is therefore at least $\frac{(\delta-1)|U_{\delta-1}|+b}{b \cdot |U_{\delta-1}|}$, which implies $\Delta_i^* > \frac{\delta-1}{b}$. Using $\delta \geq \Delta_i - \lceil \log_b n \rceil$ we get $\Delta_i \leq b \cdot \Delta_i^* + \lceil \log_b n \rceil$. □

In the following, we show an upper bound of $O(n^{1+1/\ln b})$ to produce a set of solutions with the desired approximation quality.

Lemma 11.1.3 *The total number of local improvements until a minimum spanning forest of degree at most $b \cdot \Delta_i^* + \lceil \log_b n \rceil$ has been computed for each i, $1 \leq i \leq n$, can be bounded by $O(n^{1+1/\ln b})$.*

Proof: Consider a situation where a minimum spanning forest with j connected components and degree at most $b \cdot \Delta_j^* + \lceil \log_b n \rceil$ has been computed for each j, $i \leq j \leq n$. We want to show that no more than $3 \cdot (n - i) \cdot \mu$ local improvements are necessary to reach this state, where $\mu = O(n^{1/\ln b})$. Setting $i = 1$ then proves the lemma.

Let a potential function be defined as $p(s) := \sum_{j=0}^{\lceil \log_b n \rceil + 1} d_{r+j}(s) \cdot e^j$, where $r = \max\{\Delta(s_i) - \lceil \log_b n \rceil, 0\}$. The empty edge set $s_n = \emptyset$ is obviously a minimum spanning forest with n components and minimum degree, and no improvements are necessary to reach this. In addition $p(s_n) = 0$ holds. For going from i to $i - 1$ we introduce into s_i a lightest edge e that does not create a cycle. This yields a minimum spanning forest with $i - 1$ components, denoted as s_{i-1}'. Introducing an arbitrary edge into s_i increases the potential value $p(s_i)$ by at most $2e^{\lceil \log_b n \rceil + 1} \leq 2e^2 \cdot e^{\log_b n} = 2e^2 \cdot e^{\ln(n)/\ln(b)} =: \mu$, hence $p(s_{i-1}') - p(s_i) \leq \mu$, where $\mu = O(n^{1/\ln b})$. Now, s_{i-1}' can undergo a number ν_{i-1} of local improvements to arrive at a new solution s_{i-1} that is locally optimal or satisfies

1. Let $i := n$, $s_i := \emptyset$, $S := \{s_i\}$.
2. Create s_{i-1} by introducing into s_i the lightest edge that does not create a cycle.
3. Improve the solution s_{i-1} until it is locally optimal or $\Delta(s_{i-1}) = \Delta(s_i)$ holds.
4. $S := S \cup \{s_{i-1}\}$
5. $i := i - 1$
6. If $i > 1$ continue at 2., otherwise output S and stop.

Figure 11.1: Minimum Spanning Forest Optimizer (MSFO)

$\Delta(s_{i-1}) = \Delta(s_i)$, which can be achieved by reducing the whole potential $p(s'_{i-1})$. Due to Lemma 11.1.1 and 11.1.2 in both cases the claimed approximation holds.

In a local improvement $d_k(s)$ decreases by at least 1 for some $k \geq r + 2$. The potential reduces by the smallest amount if $d_k(s)$ reduces by one, $d_{k-1}(s)$ increases by three and $d_{k-2}(s)$ decreases by 2. This means that one local improvement step reduces the potential by at least $e^2 - 3e + 2 > 1/3$, i.e., a constant amount.

Therefore, and because r cannot increase in this process, the relation $p(s_i) \leq p(s_{i+1}) - \nu_i/3 + \mu$ holds for each i, $1 \leq i \leq n-1$. Using this, the potential value of a solution s'_i that has been created by the introduction of a new edge into s_{i+1} can be bounded with respect to the cumulated number of all ν_j, $i < j \leq n$, previous improvement steps by $p(s'_i) \leq (n-i)\mu - \sum_{j=i+1}^{n} \nu_j/3$. As $\nu_i \leq 3p(s'_i)$, $\sum_{j=i}^{n} \nu_j \leq 3\mu(n-i)$. □

In Lemma 11.1.3 we have shown that the number of improvements in the multi-objective model can be bounded by $O(n^{1+1/\ln b})$, which is by a factor n smaller than then the number of improvements in the algorithm of Fischer. Based on this observation we give a deterministic algorithm that computes for each i a minimum spanning forest with i connected components and degree at most $b \cdot \Delta_i^* + \lceil \log_b n \rceil$ in time $O(n^{3+1/\ln b})$ for any $b > 1$.

Let s_i be a minimum spanning forest of degree at most $b \cdot \Delta_i^* + \lceil \log_b n \rceil$. Then we can produce a minimum spanning forest s_{i-1} with $i-1$ connected components by introducing a lightest edge that does not create a cycle. If $\Delta(s_{i-1}) = \Delta(s_i)$ holds, s_{i-1} has the desired approximation quality. Otherwise we have to improve s_{i-1}. The pseudo-code of our algorithm called Minimum Spanning Forest Optimizer (MSFO) is given in Figure 11.1.

MSFO can be seen as a variant of Kruskal's algorithm where after each insertion of an edge the degree of the current solution s_{i-1} is improved as long as we cannot guarantee the desired approximation quality. The algorithm of Kruskal can be implemented in time $O((m+n)\log n)$. Hence, to bound the runtime of MSFO it is necessary to bound the number of local improvements (as done in Lemma 11.1.3) and the time to achieve such an improvement.

Lemma 11.1.4 *Let s_i be a solution with i connected components that is not locally optimal. Then an improvement can be found in time $O(n^2)$.*

Proof: There are two possibilities to improve a solution s_i. Let F be the corresponding minimum spanning forest. In the first case, the introduced edge connects two components of F and an edge from the forest has to be removed. In the second case, the improvement is achieved by introducing an edge e that creates a cycle in F. Then an edge from this cycle has to be deleted to create a new spanning forest with i connected components.

Let $W_1, \ldots, W_k$ be the distinct weight classes for which there are edges in F. For the first case, consider the edges of W_j for each j, $1 \leq j \leq k$, in F and compute the edge that reduces the value of the potential function of Lemma 11.1.3 by the largest amount. This computation can be done in time $O(n)$ for all weight classes because the computation of the reduction for a single edge e can be done in constant time and there are at most $n-2$ edges to consider. Let $g^j_{\min}$ be the smallest value that can be obtained by removing an edge e_j of weight class W_j from F. Now we consider each edge $e' \in E \setminus F$ of weight class W_j and introduce e' if the result improves s_i. We consider in the process each edge at most once and the computation of the desired potential difference can be implemented in constant time. Hence, an improvement can be found in time $O(n^2)$ in this case.

For the second case we use the idea of Fischer and investigate a depth first search traversal of the forest F represented by s_i from every vertex $v \in V$. Let w be the current vertex of the traversal and P_w be the set of edges on the path from v to w. We assign variables $M_1, \ldots, M_k$ such that $M_j, 1 \leq j \leq k$, denotes the maximum degree of those vertices adjacent to edges of weight class W_j in P_w. For a depth first search traversal we can compute the M_i variables in constant time per step using stacks. If there is an edge $\{v, w\} \in E$, let w_i be its weight. If M_i is at least two greater than the degree of v and w, and M_i is at least $\Delta(s_i) - \lceil \log_b n \rceil$, then adding $\{v, w\}$ to s_i and deleting some edge from P_w of weight class W_i adjacent to a vertex of degree M_i constitutes an improvement. The computation of the n depth first search traversals can be carried out in time $O(n^2)$ which completes the proof. □

Using the bound on the number of necessary improvements and the time bound to achieve such an improvement, we can give an upper bound on the runtime of MSFO.

Theorem 11.1.5 *The algorithm MSFO computes for any $b > 1$ in time $O(n^{3+1/\ln b})$ a set of solutions that includes for each i, $1 \leq i \leq n$, a minimum spanning forest with i connected components and degree at most $b \cdot \Delta^*_i + \lceil \log_b n \rceil$.*

Proof: Consider the time the solutions $\{s_n, s_{n-1}, \ldots, s_i\} \subset S$ have been produced. These solutions have the following properties. Each s_j, $i \leq j \leq n$, is a minimum spanning forest with j connected components. In addition, s_j is locally optimal or $\Delta(s_j) = \Delta(s_{j+1})$ holds. Obviously, s_n is a locally optimal solution. We introduce into s_i an edge e of minimal weight that does not create a cycle. This can be easily done by checking each remaining edge in time $O(m)$. Note that the whole computation in step 2 in the run of the algorithm

can be implemented in time $O((m+n)\log n)$ using the ideas of Kruskal's algorithm. After step 3, the solution s_{i-1} has minimal weight among all solutions with $i-1$ components. If e is not incident to at least one edge of degree $\Delta(s_i)$, $\Delta(s_{i-1}) = \Delta(s_i)$ holds and s_{i-1} is a solution with the desired approximation quality due to Lemma 11.1.1. Otherwise, the number of vertices with degree $\Delta(s_i)+1$ is at most 2 and we have to improve s_{i-1} to reach a locally optimal solution or to achieve $\Delta(s_{i-1}) = \Delta(s_i)$.

The number of local improvements in the run of MSFO is $O(n^{1+1/\ln b})$ as shown in Lemma 11.1.3 and an improvement of a non locally optimal solution can be found in time $O(n^2)$ due to Lemma 11.1.4. Hence, the time until MSFO has achieved the desired approximation can be bounded by $O(n^{3+1/\ln b})$. □

11.2 Nonuniform degree bounds

Könemann and Ravi (2003) have examined the case of non-uniform degree bounds B_v for all vertices $v \in V$. They presented an algorithm that finds, in time $O(n^6 \log n)$, a spanning tree where the degree of each vertex is $O(B_v + \log n)$ and whose total edge weight is at most a constant times the weight of any tree that satisfies the degree constraints. Here the assumption that there exists a tree obeying the given degree bounds is necessary. Note, that the problem to decide this question is already NP-complete. The algorithm uses a combination of primal-dual methods (see e. g. Chapter 4 in Hochbaum (1997)) and local search, where in each local search step the normalized degree of the high-degree vertices in a current spanning tree is reduced. We generalize the primal-dual idea of Könemann and Ravi to the approximation of minimum spanning forests with nonuniform degree bounds. The task is to find for each i, $1 \leq i \leq n$, a spanning forest with i connected components and minimum total edge weight such that the maximum degree of each vertex v is at most B_v. The algorithm presented here runs in time $O(n^{3+2/\ln b})$, $b > 1$ an arbitrary constant, and outputs for each i, $1 \leq i \leq n$, a spanning forest F_i of i connected components whose vertex degrees are $O(B_v + \log n)$ and whose total weight is at most a constant times the total weight of any minimum spanning forest with i connected components. Here we do not need the assumption that there exists a spanning tree obeying the given degree bounds. For each value of i where a spanning forest respecting the degree constraints exists a solution with the stated approximation quality is produced.

We first adapt some results of Könemann and Ravi to the case of spanning forests. A feasible partition of V is a set $\pi = \{V_1, \ldots, V_k\}$ where $V_i \cap V_j = \emptyset$ for all $i \neq j$, $V = V_1 \cup \ldots \cup V_k$, and the induced subgraphs $G[V_i]$ are connected. Let G_π be the graph obtained from G by contracting each V_i into a single vertex, Π be the set of all feasible partitions of V, and $x(e)$ be the variable indicating whether edge e is included in the current solution, i.e., $x(e_i) = x_i$. We consider the following integer linear program (IP) formulation for the problem of computing the minimum spanning forest with i, $1 \leq i \leq n$, connected components that obeys all degree bounds B_v.

$$\min \quad \sum_{e\in E} w(e)x(e) \tag{11.2}$$

$$\text{s.t.} \quad \sum_{e\in E[G_\pi]} x(e) \geq |\pi| - i \quad \text{for all } \pi \in \Pi \tag{11.3}$$

$$\sum_{e\in E:v\in e} x(e) \leq B_v \quad \text{for all } v \in V \tag{11.4}$$

$$x(e) \in \{0,1\} \quad \text{for all } e \in E \tag{11.5}$$

The dual of the linear programming relaxation (LP) of (IP) is given by

$$\max \quad \sum_{\pi\in\Pi}(|\pi| - i)\cdot y_\pi - \sum_{v\in V} \lambda_v B_v \tag{11.6}$$

$$\text{s.t.} \quad \sum_{\pi:e\in E[G_\pi]} y_\pi \leq w(e) + \lambda_u + \lambda_v \quad \text{for all } e = \{u,v\} \in E \tag{11.7}$$

$$y, \lambda \geq 0 \tag{11.8}$$

Könemann and Ravi have given a primal-dual interpretation of Kruskal's algorithm. Let (IP-SP) denote (IP) without constraints of type (11.4) its LP relaxation denoted by (LP-SP) and its dual be (D-SP). Kruskal's algorithm can be seen as a continuous process over time that starts with an empty edge set at time 0 and ends with a minimum spanning tree at time t^*. At any time t, $0 \leq t \leq t^*$, a pair (x^t, y^t) is kept, where x^t is a partial primal solution for (LP-SP) and y^t is feasible solution for (D-SP). In the initialization step $x(e)^0 = 0$ is set for all $e \in E$, and $y^t_\pi = 0$ for all $\pi \in \Pi$. Consider the forest F^t that corresponds to the partial solution x^t and let π^t be the partition induced by the connected components of $G[F^t]$. At time t the algorithm increases y^t_π until a constraint of type (11.7) becomes tight. If this happens for edge e, this edge e is included into the primal solution. If more than one edge becomes tight, the edges are processed in arbitrary order. We denote by MSF_i a variant of this algorithm that stops when a minimum spanning forest with i connected components has been computed in the continuous process.

Let $deg_F(v)$ be the degree of vertex v in the spanning forest F with i connected components. The normalized degree of a vertex v is denoted by $ndeg_F(v) = \max\{0, deg_F(v) - 1 - b\alpha \cdot B_v\}$, where b and α are constants depending on the desired approximation quality. Let Δ^t the maximum normalized degree of any vertex in the current spanning forest F^t_i at a given time t and denote by U^t_j the set of vertices whose normalized degree is at least j at time t. The following lemma was shown by Könemann and Ravi (2003). It is crucial to show the approximation quality, so we present the proof.

Lemma 11.2.1 *There is a* $d^t \in \{\Delta^t - 2\log_b n, \ldots, \Delta^t\}$ *such that*

$$\sum_{v\in U_{d^t-1}} B_v \leq b \cdot \sum_{v\in U_{d^t}} B_v$$

1. $t := 0$; set $\lambda_v^t := 0$ for all $v \in V$, set $w^t(e) = w(e)$ for all $e \in E$;
2. $i := n$; $(x^t, y^t) := \mathrm{MSF}_i(G, w^t)$; $S := \{x^t\}$;
3. while $i > 1$ do
   (a) $i := i - 1$; $w^{t+1}(e) = w^t(e)$; $(x^{t+1}, y^{t+1}) := \mathrm{MSF}_i(G, w^{t+1})$; $t := t + 1$;
   (b) while $\Delta^t > 2 \log_b n$ do
      i. Choose $d^t \in \{\Delta^t - 2\log_b n, \ldots, \Delta^t\}$ s.t. $\sum_{v \in U_{d^t-1}} B_v \leq b \cdot \sum_{v \in U_{d^t}} B_v$
      ii. Choose ϵ^t and let $\lambda_v^{t+1} := \lambda_v^t + \epsilon^t$ if $v \in U_{d^t-1}^t$ and $\lambda_v^{t+1} := \lambda_v^t$ otherwise
      iii. $w^{t+1}(e) := w^t(s) + \epsilon^t$ if $((e \in F_i^t) \wedge (e \cap U_{d^t} \neq \emptyset) \vee ((e \notin F_i^t) \wedge (e \cap U_{d^t-1} \neq \emptyset))$ and $w^{t+1}(e) := w^t(e)$ otherwise
      iv. $(x^{t+1}, y^{t+1}) := \mathrm{MSF}_i(G, w^{t+1})$; $t := t + 1$;
   (c) $S := S \cup \{x^t\}$;

Figure 11.2: Primal Dual Forest Optimizer (PDFO)

for any constant $b > 1$.

Proof: Suppose that for all $d^t \in \{\Delta^t - 2\log_b n, \ldots, \Delta^t\}$ the relation

$$\sum_{v \in U_{d^t-1}} B_v > b \cdot \sum_{v \in U_{d^t}} B_v$$

holds. We may assume $B_v \leq n - 1$ for any v, which implies $\sum_{v \in V} B_v \leq n(n-1)$. Since there is at least one vertex of normalized degree Δ^t, we have

$$\sum_{v \in U_{\Delta^t - 2\log_b n}} B_v \geq b^{2\log_b n} = n^2,$$

a contradiction. □

Our algorithm called Primal Dual Forest Optimizer (PDFO) is given in Figure 11.2. The idea of the algorithm is to start with an empty edge set and compute the solutions with the desired approximation quality one after another. If we are considering a solution x^t with i connected components that does not have the desired approximation quality with respect to the degree bounds, we compute a new solution x^{t+1} which improves x^t with respect to the normalized degree. Let F_i^t be the forest corresponding to x^t. We increase the weight of an edge $e \in E$ by ϵ^t if it is either in F_i^t and adjacent to vertices of U_{d^t}, or in $E \setminus F_i^t$ and adjacent to vertices of U_{d^t-1}. The weight increment ϵ^t is defined as the smallest weight increase when deleting an edge adjacent to a vertex of U_{d^t} and inserting an edge adjacent to vertices that are not contained in U_{d^t-1} such that a new cycle-free subgraph of G

is constructed. After that, x^{t+1} is a minimum spanning forest with i connected components with respect to the updated weight function w^{t+1}. We have also stated the computation of the dual variables corresponding to the primal solutions in Figure 11.2 using the algorithm $\mathrm{MSF_i}$. The dual variables will be used later to show the approximation quality of our algorithm, but it is not necessary to carry out the computation of these variables in the run of the algorithm.

We want to show that the algorithm computes in time $O(n^{3+2/\ln b})$ an approximation of the set of minimal elements that contains for each i, $1 \leq i \leq n$, a spanning forest with i connected components in which each vertex v has degree $O(B_v + \log n)$ and weight at most a constant times the weight of an optimal solution obeying the degree bounds. First, we consider the approximation quality of the solutions that are introduced into the set S in step 3c. Here we use an extension of the arguments given by Könemann and Ravi to the case of minimum spanning forest with given degree bounds.

Lemma 11.2.2 *For all iterations $t \geq 0$ where we are considering solutions with i connected components in the algorithm PDFO, the relation*

$$\sum_{\pi \in \Pi} (|\pi| - j) y_\pi^{t+1} \geq \sum_{\pi \in \Pi} (|\pi| - j) y_\pi^t + \epsilon^t \alpha \sum_{v \in U_{d^t - 1}} B_v \tag{11.9}$$

holds for and all j, $1 \leq j \leq i$.

Proof: Let $F_j^t = \{e_1^t, \ldots, e_{n-j}^t\}$ be the set of edges that would be produced by a run of the algorithm MSF_j in iteration t. The change of the dual objective function value in iteration t for a specific value of j is given by

$$\sum_{\pi \in \Pi} (|\pi| - j) \cdot (y_\pi^{t+1} - y_\pi^t) = \sum_{l=1}^{n-j} (r_l^{t+1} - r_l^t)$$

where r_l^t is the time at which the MSF_j algorithm includes the edge e_l^t. Assume that we are considering solutions with i connected components in iteration t. Then we lengthen all edges $e \in F_i^t$ that are incident to vertices of normalized degree at least d^t. This implies that all these edges become tight ϵ^t time later. Using that all edges of F_i^t are also contained in each minimum spanning forest F_j^t for $j \leq i$, we get

$$\sum_{\pi \in \Pi} (|\pi| - j) \cdot (y_\pi^{t+1} - y_\pi^t) \geq \epsilon^t \cdot |E(U_{d^t}) \cap F_i^t|.$$

Here $E(U_{d^t})$ denotes the set of edges that are incident to vertices from U_{d^t}. F_i^t is a minimum spanning forest with i connected components. This implies that there are at most $|U_{d^t}| - i$ edges in $E(U_{d^t})$ that are incident to two vertices from U_{d^t}, therefore

$$\epsilon^t \cdot |E(U_{d^t}) \cap F_i^t| \;\geq\; \epsilon^t \cdot \left[\left(\sum_{v \in U_{d^t}} (b\alpha + 1/B_v) \cdot B_v \right) - (|U_{d^t}| - i) \right].$$

This leads to

$$\sum_{\pi\in\Pi}(|\pi|-j)(y_\pi^{t+1}-y_\pi^t) \;\geq\; \epsilon^t\alpha b\cdot\left(\sum_{v\in U_{d^t}} B_v\right)+i \geq \epsilon^t\alpha b\cdot\sum_{v\in U_{d^t}} B_v,$$

and using Lemma 11.2.1 we get

$$\sum_{\pi\in\Pi}(|\pi|-j)(y_\pi^{t+1}-y_\pi^t) \;\geq\; \epsilon^t\alpha\cdot\sum_{v\in U_{d^{t-1}}} B_v,$$

which completes the proof. □

Lemma 11.2.3 *Let $\omega > 1$ be a constant and $\alpha = \max\{\omega/(\omega-1),\omega\}$. For all iterations $t \geq 0$ where we are considering solutions with i connected components in the algorithm PDFO, the relation*

$$\omega\sum_{v\in V} B_v\lambda_v^t \;\leq\; (\omega-1)\sum_{\pi\in\Pi}(|\pi|-j)\cdot y_\pi^t \tag{11.10}$$

holds for each j, $1 \leq j \leq i$.

Proof: After initialization, $\sum_{v\in V} B_v\lambda_v^0 = \sum_{\pi\in\Pi}(|\pi|-j)\cdot y_\pi^0 = 0$ holds. Lemma 11.2.2 implies that the right hand side of (11.10) increases by at least $(\omega-1)\cdot\alpha\epsilon^t\sum_{v\in U^t_{d^t-1}} B_v$. The left hand side increases by $\omega\cdot\epsilon^t\sum_{v\in U^t_{d^t-1}} B_v$. Using $\alpha \geq \omega/(\omega-1)$, the relation is maintained. □

Lemma 11.2.4 *For all iterations $t \geq 0$ where we are considering solutions with i connected components in the algorithm PDFO, the relation*

$$\sum_{e\in F_j^t} w(e) \;\leq\; \omega\left[\sum_{\pi\in\Pi}((|\pi|-j)\cdot y_\pi^t)-\sum_{v\in V}(B_v\cdot\lambda_v^t)\right] \tag{11.11}$$

holds for each j, $1 \leq j \leq i$.

Proof: For $t = 0$ this is obviously true. Let F_j^t be the spanning forest produced by the algorithm MSF_j in the t-th iteration and let $w^t(F_j^t)$ be the weight of this spanning forest with respect to the weight function w^t. As the weights can only increase during the run of PDFO,

$$w(F_j^t) \leq w^t(F_j^t) = \sum_{e\in F_j^t} w^t(e) = \sum_{\pi\in\Pi}(|\pi|-j)\cdot y_\pi^t$$

holds. Using the invariant given in Lemma 11.2.3 we get

$$w(F_j^t) \;\leq\; \omega\left[\sum_{\pi\in\Pi}((|\pi|-j)\cdot y_\pi^t)-\sum_{v\in V}(B_v\cdot\lambda_v^t)\right]$$

□

Lemma 11.2.4 shows that in each iteration the weight of a spanning forest with j, $1 \leq j \leq i$, is only a constant times the weight of an optimal solution. It remains to show an upper bound on the runtime of PDFO. To do this we first consider the time to produce a new solution x^{t+1} from the current solution x^t.

Lemma 11.2.5 *The solution x^{t+1} can be computed from x^t in time $O(n^2)$.*

Proof: If the computation of x^{t+1} is carried out in step 3a of the algorithm introducing the lightest edge for the weight function w^{t+1} into x^t that does not create a cycle yields x^{t+1}. This can be done in time $O(n^2)$ by inspecting every edge at most once. In the other case x^t is a minimum spanning forest with i connected components with respect to w^t and x^{t+1} is a minimum spanning forest with i connected components with respect to the updated weight function w^{t+1}. To determine x^{t+1} we have to compute ϵ^t and execute the resulting exchange operation. For the computation of d^t we use an integer array of size n and store at position j, $0 \leq j \leq n-1$, the sum over the B_v-values with vertices of normalized degree j in F_i^t. This can be done in time $O(n)$ using a breath first search traversal on F_i^t in which we compute the B_v value for the current vertex v in the traversal and add the value to the value of the corresponding position in the array. After that we determine the values $\sum_{v \in U_j} B_v$, $0 \leq j \leq n-1$, one after another starting with U_0. Note that $\sum_{v \in U_0} B_v = \sum_{v \in V} B_v$. The value $\sum_{v \in U_{j+1}} B_v$ can be computed by subtracting from $\sum_{v \in U_j} B_v$ the entry at position j in the array. Each computation can be done in constant time based on the corresponding array values. Hence, the value d^t due to Lemma 11.2.1 can be determined in time $O(n)$. To compute the ϵ^t value we determine the exchange operation that leads to a primal solution of $\mathrm{MSF}_i(G, w^{t+1})$. Note that ϵ^t is the smallest weight increase such that deleting an edge adjacent to at least one vertex of U_{d^t} and inserting an edge adjacent to vertices that are not contained in U_{d^t-1} yields a minimum spanning forest with i components for the weight function w^{t+1}. We consider two possibilities for the exchange operation.

First we investigate the case where introducing an edge e connects two components of the current forest F_i^t. Then another edge from the resulting forest has to be removed to create a solution with i connected components. Introducing the edge e with smallest weight that is not incident to vertices of U_{d^t-1} and deleting the edge $e' \in F_i^t$ that has the largest weight of all edges incident to vertices of U_{d^t} in F_i^t leads to the desired exchange operation with the smallest weight increase. Each edge of G has to be examined once, which gives an upper bound of $O(n^2)$ on the runtime in this case. Let $\epsilon^{t'}$ be the value obtained by this exchange operation.

The other possibility to get a smaller value than $\epsilon^{t'}$ is to introduce into F_i^t an edge that creates a cycle. Then we have to delete one edge of this cycle. We use a depth first search traversal of F_i^t from every vertex $v \in V$. Let w be the current vertex in this traversal. Assume that there is an edge $e = \{v, w\}$ in E and that v and w are not contained in U_{d^t-1}. Otherwise we can continue the traversal since the pair $\{v, w\}$ does not fulfill the properties for the exchange operation. Let w_i be the largest weight of an edge e' in the path from v to w that is incident to vertices of U_{d^t}. If no such edge edge e' exists in the path from v to w, e can not participate in the exchange operation we are looking for.

The weight increase of introducing e and deleting e' can be computed in constant time, and the w_i variables can be maintained in constant time per step of the traversal using stacks. Hence, we can determine the exchange operation with the smallest weight increase in the second case in time $O(n^2)$. Let $\epsilon^{t''}$ be weight increase of this exchange operation. Choosing $\epsilon^t = \min\{\epsilon^{t'}, \epsilon^{t''}\}$ and computing a primal solution of MSF_i^{t+1} by executing the corresponding exchange operation gives the stated upper bound. In addition we update the weight with respect to w^{t+1} for the next iteration which can be done in time $O(n^2)$. □

Now we can prove the main result of this section and show the approximation quality and an upper bound on the runtime of the algorithm PDFO.

Theorem 11.2.6 *The algorithm PDFO computes for any $b > 1$ and $\omega > 1$ in time $O(n^{3+2/\ln b})$ a set of solutions that includes for each i, $1 \leq i \leq n$, a minimum spanning forest with i connected components in which each vertex degree is at most $b \cdot \alpha \cdot B_v + 2\log_b n + 1$ and the total weight is at most $\omega \cdot w(F_i^*)$, where $\alpha = \max\{\omega/(\omega-1), \omega\}$ and $w(F_i^*)$ is the minimum weight of any spanning forest with i connected components satisfying the degree bounds.*

Proof: As long as the algorithm has not achieved a solution with i connected components such that the vertex degree is at most $b \cdot \alpha \cdot B_v + 2\log_b n + 1$, the right hand side of (11.11) is ω times the optimal value of the dual objective function. This implies that the weight of a minimum spanning forest with i connected components for the weight function w^t is at most ω times the value of an optimal solution obeying the degree bounds for each j, $1 \leq j \leq i$. Hence, the solutions introduced into the set S in step 3c of PDFO have the stated approximation quality. As each possible value of i is considered in the run of PDFO, the set S includes after termination for each i, $1 \leq i \leq n$, a solution with i connected components that has the desired approximation quality.

In the following we give an upper bound of $O(n^{3+2/\ln b})$ on the runtime until the algorithm terminates. A new solution x^{t+1} can be computed from the current solution in time $O(n^2)$ due to Lemma 11.2.5. It remains to bound the number of primal solutions x^t that have to be computed until the algorithm terminates. The number of solutions that are computed in step 3a of the algorithm is at most $n-1$ as the number of connected components is bound by n. In the inner while-loop we compute for the solution with i connected components new primal solutions as long as we have not reached a solution with i connected components that has the desired approximation quality.

We consider a modification of the potential function p introduced in Lemma 11.1.3. Consider a solution s, let $\hat{\Delta}(s)$ be the maximum normalized degree of s, and let $\hat{d}_i$, $0 \leq i \leq n-1$, be the number of vertices with normalized degree i in this solution. The potential of a solution s is given by $p'(s) := \sum_{j=0}^{\lceil 2\log_b n\rceil+1} \hat{d}_{r+j}(s) \cdot e^j$, where $r = \hat{\Delta}(s) - \lceil 2\log_b n\rceil$. Assume the a minimum spanning forest with i, $2 \leq i \leq n$, has been computed. After initialization this is true for $i = n$. The solution of $\mathrm{MSF}_{i-1}(G, w^t)$ differs from $\mathrm{MSF}_i(G, w^t)$ by one single edge that is additionally introduced into $\mathrm{MSF}_{i-1}(G, w^t)$ at any time t. Introducing this

edge into s_i, a solution s'_{i-1} with $p'(s'_{i-1}) - p'(s_i) = O(n^{2/\ln b})$ is created. Each iteration of the inner while-loop reduces the potential by at least 1/3. Hence, we can upper bound the number of improvements in the run of PDFO by $O(n^{1+2/\ln b})$ using the ideas of Lemma 11.1.3. □

Note that choosing b as a constant large enough the runtime of PDFO approximates the upper bound $O(n^3)$. For $b = e$, $b = e^2$, ..., $b = e^k$, where $e = 2.71...$ and k is a constant, we get runtimes $O(n^5)$, $O(n^4)$, ..., $O(n^{3+2/k})$. The degrees of the produced solutions are bounded by $O(B_v + \log n)$, and the weight of a solution with i connected components introduced into the set S in step 3c is at most a constant times the weight of any minimum spanning forest with i connected components obeying the degree bounds.

11.3 Conclusions

In this chapter we have shown that a multi-objective formulation can help to design faster approximation algorithms for the generalization of two NP-hard spanning tree problems. Our algorithms can be seen as an incremental construction procedure starting with the empty edge set and producing the solutions for the different number of connected components after another. Nevertheless we point out that our results have been obtained by the multi-objective formulation and this new view on the problem. We also think that this approach may be helpful to get a better understanding of other problems that have additional constraints. Based on our observations we have given an algorithm that computes for each i, $1 \leq i \leq n$, a minimum spanning forest with i components and degree at most $b \cdot \Delta_i^* + \lceil \log_b n \rceil$ in a total time of $O(n^{3+1/\ln b})$. In the case of nonuniform degree bounds we have presented an algorithm that runs in time $O(n^{3+2/\ln b})$ and computes for each i, $1 \leq i \leq n$, a spanning forest in which each vertex has degree $O(B_v + \log n)$ and the weight is a most a constant times the weight of a minimum spanning forest with i components obeying the given degree bounds.

A conference version that contains the results of this chapter has been published in the Proceedings of the Latin American Theoretical Informatics Symposium (LATIN) 2006 (see Neumann and Laumanns (2006)). A journal version has been submitted for publication.

Chapter 12

Summary and Future Work

In this thesis we have examined the runtime behavior of randomized search heuristics on different combinatorial optimization problems. Most of the time we have considered evolutionary algorithms. This kind of randomized search heuristic has widely been applied to combinatorial optimization problems as well as to complex engineering problems. In the case of complex engineering problems often the structure of the given problem is not known and the quality of a certain parameter setting can only be evaluated by experiments of simulations. An analysis of randomized search heuristics has to consider well-known problems as otherwise it is not possible to say anything about the runtime behavior. There is no hope that such general purpose heuristics can outperform specialized algorithms on the considered problems. Nevertheless our analyses give new insights on the behavior of these heuristics that may help to come up with better randomized search heuristics for new problems where no specialized algorithms are known.

For the Eulerian cycle problem, we have shown that evolutionary algorithms are able to compute an Eulerian cycle of a given Eulerian graph in expected polynomial time if jumps are used for mutation. Switching from jumps to exchanges there are instances where the expected optimization time change drastically, i. e. from polynomial to exponential. In the case of the minimum spanning tree problem, evolutionary algorithms have to cope with the possibility that there are exponentially many spanning trees that have different weights. Nevertheless, they are able to compute a minimum spanning tree in expected polynomial time. Examining the minimum spanning tree problem we have also shown that sometimes a multi-objective model of a given single-objective optimization problem can direct the search of an evolutionary algorithm in a better way. We have given runtime bounds for the corresponding MOEAs that are smaller than the lower bounds obtained for their single-objective counterparts. In addition our experimental studies shown the advantage of the multi-objective model for the case of dense random graphs.

For the multi-objective minimum spanning tree problem, we have shown that the extremal points of the Pareto front constitute a 2-approximation of the Pareto set. In addition we have proven that simple MOEAs are able to compute a set of solutions which contains

for each extremal point a corresponding solution in expected pseudo-polynomial time. To our knowledge this is the first rigorous result on the runtime behavior of a MOEA on an NP-hard multi-objective combinatorial optimization problem.

Another important class of randomized search heuristic is ant colony optimization. We have shown that a simple algorithm belonging to this class behaves for a certain parameter setting as a simple evolutionary algorithm for which many results on the runtime behavior are known. Hence, these results, including the ones presented in Chapter 7, transfer to the considered ACO algorithm. In addition the behavior of the ACO algorithm on the pseudo-boolean function OneMax has been investigated in greater detail and a threshold behavior has been proven for varying evaporation factor.

After we obtained the mentioned results on the behavior of randomized search heuristics, we have switched to classical approximation algorithms. Motivated by the results on the behavior of evolutionary algorithms on the minimum spanning tree problem, we have investigated whether a multi-objective model of a single-objective optimization problem can also help to come up with faster approximation algorithms. We have shown that this is indeed the case by examining the generalization of two NP-hard spanning tree problems.

The randomized search heuristics analyzed in this thesis are simple ones. For future work it would be interesting to examine the influence of a larger population for some classical combinatorial optimization problems. Also the influence of crossover operators seems to be an interesting point which should be investigated. In the case of ACO algorithms, we are at the beginning of rigorous analyses. The results presented in Chapter 8 are the first of that kind. In the future, it would be interesting to examine more pseudo-boolean functions in greater detail as well as the behavior of ACO algorithms on other construction graphs. Another interesting point is to investigate ACO algorithms that work with a larger number of ants in each iteration.

Appendix A

Mathematical Background

We present some elementary mathematical material that has been used throughout this thesis. Most of these basics in mathematics can be found in Feller (1968, 1971) or Motwani and Raghavan (1995).

A.1 Probability distributions

Definition A.1.1 ***(Normal distribution):***
A random variable X is called normally distributed with expectation μ and variance σ^2 and denoted by $\Phi(\mu, \sigma^2)$ if it has the density function

$$n(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

In the case of $\mu = 0$ and $\sigma^2 = 1$ it is called the standard normal distribution and denoted by Φ.

Definition A.1.2 ***(Binomial distribution):***
The binomial distribution with parameter n and p which is denoted by $B(n, p)$ and defined by the probability $b(k; n, p)$ that n Bernoulli trials with probabilities p for success and $q = 1-p$ for failure result in k successes and $n - k$ failures. $B(n, p)$ has the density function

$$b(k; n, p) = \binom{n}{k} p^k q^{n-k}.$$

Definition A.1.3 ***(Poisson distribution):***
Let λ be a positive real number. The Poisson distribution with parameter λ has the density function

$$p(x) = \begin{cases} \lambda^x e^{-\lambda}/x! & : \quad x = 0, 1, 2, \ldots \\ 0 & : \quad otherwise \end{cases}.$$

Proposition A.1.4 ***(Central Limit Theorem):***
Let $X_1, X_2, \ldots, X_n$ be mutually independent random variable with a common distribution D. Assume that the expectation μ and the standard deviation σ of D exist and are finite. Let $S_n = X_1 + \ldots + X_n$ then

$$\lim_{n\to\infty} Prob\left(\frac{S_n - n\mu}{\sigma\sqrt{n}} < x\right) = \Phi(x),$$

where Φ denotes the standard normal distribution.

Proposition A.1.5 ***(Lindeberg's generalization of the Central Limit Theorem):***
Let $X_1, X_2, \ldots, X_n$ be mutually independent random variables with distributions $F_1, F_2, \ldots, F_n$. Assume that

$$E(X_k) = 0, \qquad \mathrm{Var}(X_k) = \sigma_k^2$$

and define

$$s_n^2 = \sigma_1^2 + \ldots + \sigma_n^2.$$

If for all $t > 0$

$$\frac{1}{s_n^2}\sum_{k=1}^{n}\int_{|y|\geq ts_n} y^2 F_k\{dy\} \to 0 \tag{A.1}$$

holds , the distribution of the normalized sum

$$S_n^* = \frac{X_1 + \ldots + X_n}{s_n}$$

tends to the standard normal distribution Φ.

Condition (A.1) is called the Lindeberg condition. Assume that the X_k are uniformly bounded, which means that all F_k are carried by some finite interval $[-a, a]$. Then (A.1) is satisfied iff $s_n \to \infty$.

A.2 Deviation inequalities

Proposition A.2.1 ***(Markov's inequality):***
Let X be a random variable assuming only non-negative values. Then for all $t \in \mathbb{R}^+$,

$$Prob(X \geq k \cdot E(X)) \leq 1/k.$$

Proposition A.2.2 ***(Chernoff bounds):***
Let $X_1, X_2, \ldots, X_n$ be independent Poisson trials such that for $1 \leq i \leq n$ $Prob(X_i = 1) = p_i$, where $0 < p_i < 1$. Let $X = \sum_{i=1}^n X_i$, $\mu = E(X) = \sum_{i=1}^n p_i$. Then the following inequalities hold.

$$
\begin{aligned}
Prob(X > (1+\delta)\mu) &< \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu} \qquad \delta > 0 \\
Prob(X > (1+\delta)\mu) &< e^{-\mu\delta^2/3} \qquad 0 < \delta \leq 1 \\
Prob(X < (1+\delta)\mu) &< e^{-\mu\delta^2/2} \qquad 0 < \delta \leq 1
\end{aligned}
$$

The following Chernoff-Hoeffding-type bound can be found as Theorem 3.44 in Scheideler (2000).

Proposition A.2.3 ***(Chernoff-Hoeffding-type bound):***
Let $X_1, \ldots, X_n$ be be independent random variables such that $X_i - E(X_i) \leq b$, $1 \leq i \leq n$. Let $X = \sum_{i=1}^{n} X_i$ have expected value μ and variance ν, then for any $d \geq 0$,

$$
\begin{aligned}
Prob(X - \mu \geq d) &\leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\nu/b^2} \\
&\leq e^{-\frac{d^2}{2\nu(1+\delta/3)}}
\end{aligned}
$$

holds, where $\delta = b \cdot d/\nu$.

A.3 Other useful formulas

Proposition A.3.1 ***(Stirling's formula):***
For any $n \in \mathbb{N}$

$$\sqrt{2\pi n} n^n e^{-n} < n! < \sqrt{3\pi n} n^n e^{-n}$$

holds.

Proposition A.3.2 ***(Binomial coefficients):***

Let $n \geq k \geq 0$. The binomial coefficients are defined as

$$\binom{n}{k} = \binom{n}{n-k} = \frac{n!}{k!(n-k)!}$$

and it holds

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \frac{n^k}{k!}.$$

Proposition A.3.3 ***(Harmonic sum):***
Let $H_n = \sum_{i=1}^{n} 1/i$ be the nth Harmonic sum. Then for any $n \in \mathbb{N}$

$$H_n = \ln n + \Theta(1)$$

holds.

Proposition A.3.4 ***(Coupon collector's theorem):***
In the coupon collector's problem n different coupons are given and at each trial a coupon is chosen uniformly at random. Let X be a random variable describing the number of trials required to choose each coupon at least once. Then

$$E(X) = nH_n$$

holds, where H_n denotes the nth Harmonic number, and

$$\lim_{n\to\infty} \text{Prob}(X \leq n(\ln n - c)) = e^{-e^c}$$

holds for each constant $c \in \mathbb{R}$.

Appendix B

ACO Algorithms

We repeat Hoeffding's technique, leading to Lemma B.1.4. Note that the following statements constitute only minor modifications of the first pages in Hoeffding (1956).

B.1 Modifications of the Hoeffding Lemma

The expected value of a function $g(S)$ is

$$f(p) = E(g(S)) = \sum_{k=0}^{n} g(k) A_{nk}(p), \tag{B.1}$$

where $p = (p_1, \ldots, p_n)$ and the probability A_{nk} of $S = k$ is given by

$$A_{nk}(p) = \sum_{\substack{(i_1,\ldots,i_n)\in\{0,1\}^n,\\ i_1+\cdots+i_n=k}} \prod_{j=1}^{n} p_j^{i_j}(1-p_j)^{1-i_j}, \quad k = 0, 1 \ldots, n.$$

The function $f(p)$ is symmetric in the components of p and linear in each component. Any function with these two properties can be represented in form (B.1). We consider the problem of finding the maximum and the minimum of $f(p)$ in the section D of the hyperplane

$$p_1 + p_2 + \cdots + p_n = np \quad (1/n < p < 1 - 1/n).$$

We denote by $p^{i_1,i_2,\ldots,i_m}$ the point in the $(n-m)$-dimensional space, which is obtained from p by omitting the coordinates $p_{i_1}, p_{i_2}, \ldots, p_{i_m}$.

Since $f(p)$ is symmetric, and linear in each component, we can write

$$f(p) = f_{n-1,0}(p^j) + p_j f_{n-1,1}(p^j), \quad j = 1, 2, \ldots, n, \tag{B.2}$$

where the functions $f_{n-1,0}$ and $f_{n-1,1}$ are independent of the index j and symmetric and linear in the components of p^j.

We define the functions $f_{n-k,i}$ by $f_{n,0}(p) = f(p)$ and

$$f_{n-k,i}(p^{1,2,\ldots,k}) = f_{n-k-1,i}(p^{1,2,\ldots,k+1}) + p_{k+1} f_{n-k-1,i+1}(p^{1,2,\ldots,k}), \\ i = 0, 1, \ldots, k, \quad k = 0, 1, \ldots n-1. \quad \text{(B.3)}$$

We obtain

$$f(p) = \sum_{i=1}^{m} C_{mi}(p_1, p_2, \ldots, p_m) f_{n-m,i}(p^{1,\ldots,m}), \quad m = 1, 2, \ldots, n, \quad \text{(B.4)}$$

where $C_{m0}, C_{m1}, \ldots, C_{m,m}$ are the symmetric sums

$$C_{m0}(p_1, p_2, \ldots, p_m) = 1 \quad \text{(B.5)}$$

and

$$C_{mi}(p_1, p_2, \ldots, p_m) \\ = (p_1 p_2 \cdots p_i) + (p_1 p_2 \cdots p_{i-1} p_{i+1}) + \cdots + (p_{m-i+1} p_{m-i+2} \cdots p_m)$$

for $i > 0$.

Theorem B.1.1 *Let $a = (a_1, a_2, \ldots, a_n)$ be a point in D at which $f(p)$ attains its maximum. Then for every two distinct indices i, j, we have*

$$f_{n-2,2}(a^{ij}) \leq 0 \quad \textit{if } a_i \neq a_j, \quad \text{(B.6)}$$

$$f_{n-2,2}(a^{ij}) = 0 \quad \textit{if } a_i \neq a_j, 1/n < a_i, a_j < 1 - 1/n, \quad \text{(B.7)}$$

$$f_{n-2,2}(a^{ij}) \geq 0 \quad \textit{if } 1/n < a_i = a_j < 1 - 1/n. \quad \text{(B.8)}$$

Proof: Let a' denote the point which is obtained from a if a_i and a_j are replaced by $a_i + x$ and $a_j - x$. The point a' is in D for all x in the interval I defined by $1/n \leq a_i + x \leq 1 - 1/n$, $1/n \leq a_j - x \leq 1 - 1/n$. We have

$$f(a') = f_{n-2,0}(a^{ij}) + (a_i + a_j) f_{n-2,1}(a^{ij}) + (a_i + x)(a_j - x) f_{n-2,2}(a^{ij}).$$

Hence,

$$f(a') - f(a) = x(a_j - a_i - x) f_{n-2,2}(a^{ij}). \quad \text{(B.9)}$$

Since $f(a)$ is a maximum, the right side must be negative or zero for all x in I. We may assume $a_i \leq a_j$. If $a_i \neq a_j$, we can choose x positive and sufficiently small such that x is in I and (B.6) holds. If $1/n < a_i < 1 - 1/n$ and $1/n < a_j < 1 - 1/n$ then the point $x = -a_i + 1/n$ is in the interior of I and (B.8) must hold. Moreover, if $a_i \neq a_j$, together with (B.6), we obtain (B.7). If the maximum is not attained at a' when x is in I and is different and sufficiently close to zero, the inequalities (B.6) and (B.8) must be strict. □

In general, the maximum or minimum of $f(p)$ can be attained at more than one point in D. The following theorem gives some information about the set of points at which an extremum is attained.

Theorem B.1.2 *Let a be a point in D at which $f(p)$ attains its maximum or its minimum. Suppose that a has at least two unequal coordinates which are distinct from $1/n$ and $1-1/n$. Then $f(p)$ attains its maximum (or minimum) at any point in D which has the same number of $1/n$ coordinates and the same number of $1-1/n$ coordinates as a has.*

Proof: Let $m = n - r - s$ be the number of coordinates of $a = (a_1, \ldots, a_n)$ which are distinct from $1/n$ and $1-1/n$. We may take $a_1, \ldots, a_m$ to be these coordinates and assume $a_1 \neq a_2$. We first show

$$f_{n-k,i}(a_{k+1}, \ldots, a_n) = 0, \quad i = 2, \ldots, k. \tag{B.10}$$

We prove this equation by induction on k. Due to Theorem B.1.1 this holds for $k = 2$. Let

$$b_k = (b_1, \ldots, b_k, a_{k+1}, \ldots, a_n), \tag{B.11}$$

where

$$b_1 + \cdots + b_k = a_1 + \cdots + a_k, \quad 1/n \leq b_i \leq 1 - 1/n, \quad i = 1, \ldots, k. \tag{B.12}$$

The point b_k is in D. By (B.4) and the induction hypothesis,

$$f(b_k) = f_{n-k,0}(a_{k+1}, \ldots, a_n) + (a_1 + \cdots + a_k) f_{n-k,1}(a_{k+1}, \ldots, a_n) = f(a). \tag{B.13}$$

Thus, the maximum is attained at every point b_k which satisfies (B.11) and (B.12). In particular (B.12) can be satisfied with $b_1 \neq b_2$, $b_1 \neq a_{k+1}$, $b_2 \neq a_{k+1}$, $1/n < b_i < 1 - 1/n$, $i = 1, \ldots, k$ (since $1/n < a_j < 1 - 1/n$ for $j = 1, \ldots, m$). Under these assumptions, we can apply the induction hypothesis (B.10) with a replaced by the point b_k, whose first $k+1$ coordinates can by suitably rearranged. Hence,

$$f_{n-k,i}(b_1, a_{k+2}, \ldots, a_n) = 0, \quad f_{n-k,i}(b_2, a_{k+2}, \ldots, a_n) = 0, \quad i = 2, \ldots k.$$

Applying (B.3) to the left sides of these equations, we obtain

$$f_{n-k-1,i}(a_{k+2}, \ldots, a_n) + b_h f_{n-k-1,i+1}(a_{k+2}, \ldots, a_n) = 0, \qquad i = 2, \ldots, k, \quad h = 1, 2. \tag{B.14}$$

Since $b_1 \neq b_2$, we find that (B.10) is satisfied with k replaced by $k+1$. Thus (B.10) holds for $k = 2, \ldots, m$. Equation (B.13) holds for every b_m that satisfies (B.11) and (B.12) and f is symmetric which completes the proof. □

Corollary B.1.3 *The maximum and minimum of $f(p)$ in D are attained at points whose coordinates take on at most three different values, only one of which is distinct from $1/n$ and $1 - 1/n$.*

Lemma B.1.4 *Let $X_1, \ldots, X_k \in \{0, 1\}$ be independent Poisson trails with success probabilities $p_i \in [1/n, 1 - 1/n]$, $1 \leq i \leq k$. Let $X = X_1 + \cdots + X_k$ and $\mu = p_1 + \cdots + p_k$. Then $Prob(X \geq \mu + 1/2)$ is minimized if the p_i take on at most three different values, only one which is distinct from $1/n$ and $1 - 1/n$.*

Proof: Consider (B.1) and set $g(k) = 1$ if $k \geq \mu + 1/2$ and $g(k) = 0$ otherwise. Hence $f(p)$ computes in this case the probability of obtaining a value at least $\mu + 1/2$ and we can apply Corollary B.1.3. □

Bibliography

[1] Barahona, F., Pulleyblank, W. R. (1987). Exact arborescences , matching and cycles. Discrete Applied Mathematics, 16, 91–99.

[2] Beier, R., and Vöcking, B. (2003). Random knapsack in expected polynomial time. Proc of the ACM Symposium on Theory of Computing (STOC), 306–329.

[3] Beyer, H.-G., Schwefel, H.-P., and Wegener, I. (2002). How to analyse evolutionary algorithms. Theoretical Computer Science 287, 101–130.

[4] Briest, P., Brockhoff, D., Degener, B., Englert, M., Gunia, C., Heering, O., Jansen, T., Leifhelm, M., Plociennik, K., Röglin, H., Schweer, A., Sudholt, D., Tannenbaum, S., and Wegener, I. (2004). Experimental supplements to the theoretical analysis of EAs on problems from combinatorial optimization. In Proc. of the 8th Int. Conf. on Parallel Problem Solving from Nature (PPSN VIII). LNCS 3242, 21–30.

[5] Bui, T., Chaudhuri, S., Leighton, T., Sipser, M. (1984). Graph bisection algorithms with good average case behavior. In Proc. of the 25th IEEE Symposium onf Foundations of Computer Science (FOCS), 181–192

[6] Coello Coello, C. A., Van Veldhuizen, D. A., and Lamont, G. B. (2002). Evolutionary algorithms for solving multi-objective problems. Kluwer Academic Publishers, New York.

[7] Cohn, H., Kleinberg, R. D., Szegedy, B., and Umans, Ch. (2005). Group-theoretic algorithms for matrix multiplication. 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 379-388.

[8] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001). Introduction to algorithms. 2nd Edition, McGraw Hill, New York.

[9] Deb, K. (2001). Multi-objective optimization using evolutionary algorithms. Wiley, Chichester.

[10] Dorigo, M. and Blum, C. (2005). Ant colony optimization theory: A survey. Theoretical Computer Science 344, 243–278.

[11] Dorigo, M., Maniezzo, V., and Colorni, A. (1991). The ant system: An autocatalytic optimization process. Technical Report 91-016 Revised, Politecnico di Milano.

[12] Dorigo, M. and Stützle, T. (2004). Ant colony optimization. MIT Press.

[13] Droste, S., Jansen, T., and Wegener, I. (2002). On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science 276, 51–81.

[14] Edmonds, J. and Johnson, E.L. (1973). Matching, Euler tours and the chinese postman. Mathematical Programming, 5:88-124.

[15] Ehrgott, M. (2000). Approximation algorithms for combinatorial multicriteria optimization problems. Int. Transactions in Operational Research 7, 5–31.

[16] Feller, W. (1968). An Introduction to probability theory and its applications, vol. 1. Wiley, 3rd ed.

[17] Feller, W. (1971). An introduction to probability theory and its applications, vol. 2. Wiley, 2nd ed.

[18] Fischer, T. (1993). Optimizing the degree of minimum weight spanning trees. Technical Report 93-1338, Department of Computer Science, Cornell University, Ithaca, NY, USA.

[19] Fogel, L., Ownes, M., and Walsh, M.(1966). Artificial intelligence through simulated evolution. Wiley, New York.

[20] Fürer, M. and Raghavachari, B. (1990). An NC approximation algorithm for the minimum degree spanning tree problem. In Proc. of the 28th Annual Allerton Conf. on Communication, Control and Computing, 274-281.

[21] Fürer, M. and Raghavachari, B. (1992). Approximating the minimum-degree spanning tree to within one from the optimal degree. In Proc. of the third annual ACM-SIAM symposium on Discrete algorithms (SODA), 317-324.

[22] Fürer, M. and Raghavachari, B. (1994). Approximating the minimum-degree Steiner tree to within one of optimal. Journal of Algorithms 17, 409-423.

[23] Ehrgott, M. (2000). Multicriteria optimization. Second edition, Berlin, Springer.

[24] Gerhards, T. (2006). Evolutionary algorithms and the degree-constrained minimum spanning tree problem (in german). Master thesis, Department of Computer Science, Christian-Albrechts University of Kiel.

[25] Giel, O. (2003). Expected runtimes of a simple multi-objective evolutionary algorithm. Proc. of the Congress on Evolutionary Computation 2003 (CEC 2003), 1918–1925.

[26] Giel, O. and Wegener, I. (2003). Evolutionary algorithms and the maximum matching problem. Proc. of 20th STACS. LNCS 2607, 415–426.

[27] Giel, O. and Wegener, I. (2004). Searching randomly for maximum matchings. Electronic Colloquium on Computational Complexity (ECCC), Report No. 76.

[28] Glaser, O. (2006). Multi-objective evolutionary algorithms and the vertex cover problem (in german). Master thesis, Department of Computer Science, Christian-Albrechts University of Kiel.

[29] Gottlieb, J., Julstrom, B. A., Raidl, G. R., Rothlauf, F. (2001). Prüfer numbers: A poor representation of spanning trees for evolutionary search. In Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2001), San Francisco, CA, 343–350.

[30] Gutjahr, W. J. (2003). A generalized convergence result for the graph-based ant system metaheuristic. Probability in the Engineering and Informational Sciences 17, 545–569.

[31] Gutjahr, W. J. (2006). On the finite-time dynamics of ant colony optimization. Methodology and Computing in Applied Probability. To appear

[32] Hamacher, H.W. and Ruhe, G. (1994). On spanning tree problems with multiple objectives. Annals of Operations Research 52, 209–230.

[33] Hierholzer, C. (1873). Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. Math. Ann. 6, 30-32.

[34] Hochbaum, D. S. (1997). Approximation algorithms for NP-hard problems. PWS Publishing Company.

[35] Hoeffding, W. (1956). On the distribution of the number of successes in independent trials. Annals of Mathematical Statistics, 27, 713–721.

[36] Jansen, T. and Sudholt, D. (2005). Design and analysis of an asymmetric mutation operator. Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005). IEEE Press, Piscataway, NJ, 497–504.

[37] Jansen, T. and Wegener, I. (2001). Evolutionary algorithms - how to cope with plateaus of constant fitness and when to reject strings of the same fitness. IEEE Trans. on Evolutionary Computation 5, 589-599.

[38] Jerrum, M. and Sorkin, G.B. (1998). The Metropolis algorithm for graph bisection. Discrete Applied Mathematics 82, 155–175.

[39] Johnson, D.S., Papadimitriou, C. H., and Yannakakis, M. (1988). How easy is local search. J. Comp. Syst. Sci. 37, 79–100

[40] Kano, M. (1987). Maximum and kth maximal spanning trees of a weighted graph. Combinatorica 7, 205–214.

[41] Karger, D.R., Klein, P.N., and Tarjan, R.E. (1995). A randomized linear-time algorithm to find minimum spanning trees. Journal of the ACM 42(2): 321–328.

[42] Kehden, B. and Neumann F. (2006). A relation-algebraic view on evolutionary algorithms for some graph problems. In: Gottlieb and Raidl (Eds.): EvoCop 2006, LNCS 3906, Springer, Berlin, 147–158.

[43] Knowles, J.D. and Corne, D.W. (2001). A comparison of encodings and algorithms for multiobjective minimum spanning tree problems. Proc. of the Congress on Evolutionary Computation 2001 (CEC 2001).

[44] Könemann, J., and Ravi, R. (2003). Primal-dual meets local search: approximating MST's with nonuniform degree bounds. Proc of the ACM Symposium on Theory of Computing (STOC), 389-395.

[45] Koza, J. R. (1990). Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical report STAN-CS-90-1314, Department of Computer Science, Standford, CA.

[46] Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. of the American Mathematical Society, 7: 48–50.

[47] Lacomme, P., Prins, C., and Ramdane-Chérif, W. (2001). A genetic algorithm for the capacitated arc routing problem and its extensions. Applications of evolutionary computing: Proc. EvoWorkshops.

[48] Laumanns, M., Thiele, L. , and Zitzler, E. (2004). Running time analysis of evolutionary algorithms on a simplified multiobjective knapsack problem. Natural Computing 3, 37–51.

[49] Laumanns, M., Thiele, L., Zitzler, E., Welzl, E., and Deb, K. (2002). Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimization problem. Proc. of Parallel Problem Solving from Nature – PPSN VII. LNCS 2939, 44–53.

[50] Mayr, E.W. and Plaxton, C.G. (1992). On the spanning trees of weighted graphs. Combinatorica 12, 433–447.

[51] Michalewicz, Z. (2004). How to solve it: Modern heuristics. 2nd edition, Springer-Verlag, Berlin.

[52] Motwani, R. and Raghavan, P. (1995). Randomized algorithms. Cambridge University Press.

[53] Mühlenbein, H. (1992). How genetic algorithms really work: mutation and hillclimbing. Parallel Problem Solving from Nature - PPSN-II, Elsevier, 15–26.

[54] Nemhauser, G. and Ullmann, Z. (1969). Discrete dynamic programming and capital allocation. Management Science, 15(9), 494–505.

[55] Neumann, F. (2004a). Expected runtimes of evolutionary algorithms for the Eulerian cycle problem. Proc. of the Congress on Evolutionary Computation 2004 (CEC 2004), volume 1, IEEE Press, 904–910.

[56] Neumann F. (2004b). Expected runtimes of a simple evolutionary algorithm for the multi-objective minimum spanning tree problem. In: Yao et al. (Eds.): Parallel Problem Solving from Nature - PPSN VIII, LNCS 3242, Springer, Berlin, Germany, pages 80–89.

[57] Neumann F. (2007). Expected runtimes of a simple evolutionary algorithm for the multi-objective minimum spanning tree problem. Special Issue on Evolutionary Multi-Objective Optimization of the European Journal on Operational Research (to appear).

[58] Neumann F. and Laumanns M. (2006). Speeding up approximation algorithms for NP-hard spanning forest problems by multi-objective optimization. In: Correa et al. (Eds.): LATIN 2006, LNCS 3887, Springer, Berlin, pages 745–756.

[59] Neumann, F. and Wegener, I. (2004). Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. Deb et al. (Eds.): Genetic and Evolutionary Computation Conference - GECCO 2004, LNCS 3102, Springer, Berlin, Germany, 713–724.

[60] Neumann, F. and Wegener, I. (2005). Minimum spanning trees made easier via multi-objective optimization. In: Beyer et al. (Eds.): Genetic and Evolutionary Computation Conference - GECCO 2005, Volume 1, ACM Press, New York, USA, 763–770.

[61] Neumann, F. and Wegener, I. (2006). Minimum spanning trees made easier via multi-objective optimization. Accepted for Natural Computing.

[62] Neumann, F. and Witt, C. (2006). Runtime Analysis of a Simple Ant Colony Optimization Algorithm. Submitted for publication

[63] Papadimitriou, C. H. (1994). Computational complexity. Addison Wesley.

[64] Papadimitriou, C. H. and Steiglitz, K. (1998). Combinatorial optimization: algorithms and complexity. Dover

[65] Papadimitriou, C. H., Schäffer, A. A., and Yannakakis, M. (1990). On the complexity of local search. Proc. of 22nd ACM Symp. on Theory of Computing (STOC), 438–445.

[66] Papadimitriou, C. H. and Yannakakis, M. (2000). The complexity of tradeoffs, and optimal access of web sources. 41st Annual Symposium on Foundations of Computer Science (FOCS), 86–92.

[67] Prim, R. C. (1957). Shortest connection networks and some generalizations. Bell System Technical Journal, 36: 1389–1401.

[68] Raidl, G.R. and Julstrom, B.A. (2003). Edge sets: an effective evolutionary coding of spanning trees. IEEE Trans. on Evolutionary Computation 7, 225–239.

[69] Rechenberg, I. (1973). Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution. Frommann-Holzboog Verlag, Stuttgart, Germany.

[70] Sasaki, G. and Hajek, B. (1988). The time complexity of maximum matching by simulated annealing. Journal of the ACM 35, 387–403.

[71] Scharnow, J., Tinnefeld, K., and Wegener, I. (2002). Fitness landscapes based on sorting and shortest paths problems. Proc. of Parallel Problem Solving from Nature – PPSN VII. LNCS 2939, 54–63.

[72] Scharnow, J., Tinnefeld, K., and Wegener, I. (2004). The analysis of evolutionary algorithms on sorting and shortest paths problems. Special Issue on Evolutionary Computation in Combinatorial Optimization of the Journal of Mathematical Modeling and Algorithms, Volume 4, Issue 3, 349–366.

[73] Scheideler, C. (2000). Probabilistic methods for coordination problems. HNI-Verlagsschriftenreihe 78, University of Paderborn, Habilitation Thesis.

[74] Skiena, S. (1990). Implementing discrete mathematics: combinatorics and graph theory with mathematica. Addison-Wesley.

[75] Schwefel, H.P. (1995). Evolution and optimum seeking. Sixth-Generation Computer Technology Series, Wiley, New York.

[76] Schwefel, H.P. (1981). Numerical optimization for computer models. John Willey, Chichester, U.K..

[77] Swinscow, T. D. V. and Campbell, M. J. (2001). Statistics at square one. Bmj Publishing Group, 10th edition.

[78] Vazirani, V. V. (2003). Approximation algorithms. Springer, Second edition.

[79] Wegener, I. (2005a). Complexity theory - exploring the limits of efficient algorithms. Springer.

[80] Wegener, I. (2005b). Simulated annealing beats metropolis in combinatorial optimization. Proc. of Automata, Languages and Programming, 32nd International Colloquium (ICALP 2005), LNCS 3580, 589–601.

[81] Witt, C. (2005). Worst-case and average-case approximations by simple randomized search heuristics. Proc. of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS 2005), LNCS 3404, 44–56.

[82] Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. IEEE Trans. on Evolutionary Computation 1(1), 67–82.

[83] Zhou, G. and Gen, M. (1999). Genetic algorithm approach on multi-criteria minimum spanning tree problem. European Journal of Operational Research 114, 141–152.