Problem-specific design of metaheuristics for constrained spanning tree problems

Dissertation
zur Erlangung des Grades eines Doktors der
wirtschaftlichen Staatswissenschaften
(Dr. rer. pol.)
des Fachbereichs Rechts- und Wirtschaftswissenschaften
der Johannes Gutenberg-Universität Mainz

vorgelegt von
Dipl.-Wirt.-Inf. Wolfgang Steitz
in Mainz

im Jahre 2012

Tag der mündlichen Prüfung: 10.10.2012

Contents

List of Papers

- Wolfgang Steitz[1] and Prof. Dr. Franz Rothlauf[1] (2012). "Edge orientation and the design of problem-specific crossover operators for the OCST problem". In: *IEEE Transactions on Evolutionary Computation* 16.1, pp. 108–116
- Wolfgang Steitz and Prof. Dr. Franz Rothlauf (2012). "Using penalties instead of rewards: solving OCST problems with guided local search". In: *Swarm and Evolutionary Computation* 3, pp. 46–53
- Wolfgang Steitz and Prof. Dr. Franz Rothlauf (2009). "New insights into the OCST problem: integrating node degrees and their location in the graph". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation, Montreal, Canada*. ACM, pp. 357–364
- Wolfgang Steitz and Prof. Dr. Franz Rothlauf (2012). "Biased metaheuristics for the quadratic minimum spanning tree problem". Submitted to: *European Journal of Operational Research*.
- Wolfgang Steitz (2012). "New heuristic approaches for the bounded-diameter spanning tree problem". Submitted to: *Journal of Heuristics*.

Additional papers not included in this thesis:

- Jella Pfeiffer; Malte Probst; Wolfgang Steitz and Franz Rothlauf (2012). "Inferring decision Strategies from Clickstreams in Decision Support Systems: A New Process-Tracing Approach Using State Machines". To appear in: *Zeitschrift für Betriebswirtschaftslehre*.
- Wolfgang Steitz and Franz Rothlauf (2011). "Guided local search for the optimal communication spanning tree problem". In: *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation, Dublin, Ireland*. ACM, pp. 51–52

[1] Johannes Gutenberg-Universität Mainz, Lehrstuhl für Wirtschaftsinformatik und BWL, Jakob-Welder-Weg 9, D-55128 Mainz, Germany

- Wolfgang Steitz and Franz Rothlauf (2010). "Solving OCST problems with problem-specific guided local search". In: *Proceedings of the 12th Annual conference on Genetic and evolutionary computation, Portland, USA*. ACM, pp. 301–302
- Wolfgang Steitz and Franz Rothlauf (2008). "Orientation matters: How to efficiently solve OCST problems with problem-specific EAs". In: *Proceedings of the 10th Annual conference on Genetic and evolutionary computation, Atlanta, USA*. ACM, pp. 563–570

List of Figures

List of Tables

Chapter 1

Introduction

Many optimization tasks with practical importance are concerned with the allocation of limited resources to achieve some objective criteria. Usually the number of feasible solutions to such problems is finite, and the goal is to determine the best alternative. These optimization tasks are called *combinatorial optimization problems*. Besides prominent examples like the traveling salesman problem or vehicle routing problems, constrained spanning tree problems are an important class of problems. A spanning tree for a connected graph G is a subgraph of G that connects all nodes of G without creating any cycles. Finding good spanning trees is necessary in many areas. Whenever an economical and efficient way to connect a set of factories, IT servers, terminals, cities or others is needed, a solution is often a spanning tree. Such problems that deal with the design of optimal communication or transportation networks that satisfy a given set of constraints have been extensively studied in the literature (Wu and K.-M. Chao, 2004).

To deal with such combinatorial optimization problems, a wide range of algorithmic strategies have been proposed. Approaches can be roughly classified into three groups: exact approaches, approximation algorithms, and heuristics. Classical optimization approaches like linear programming or branch-and-bound methods yield provably optimal solutions. While some of these approaches work remarkably well, for difficult problems they need a high computational effort to generate optimal solutions and prove the optimality. Since many real-world optimization tasks are difficult (e.g. $\mathcal{NP}$-hard (Garey and Johnson, 1979)), even medium sized problem instances often become intractable for exact approaches. In such cases, one can use approximation algorithms. These are algorithms that in general do not create optimal solutions, but give provable bounds on the solution quality as well as the runtime, which is usually polynomial.

In order to solve combinatorial optimization problems that are difficult to solve, heuristics are often the best choice. While they do not necessarily identify optimal solutions or give quality guarantees, they are usually very fast (polynomial runtime), efficient and able to generate high-quality solutions for difficult and $\mathcal{NP}$-hard problems. Besides problem-specific heuristics, there exist general purpose optimization

concepts like evolutionary algorithms, ant colony optimization, simulated annealing and others, that can be easily applied to various optimization problems. These are called *metaheuristics* (see Michalewicz and Fogel, 2004 for an introduction). The runtime of metaheuristics is controlled during implementation, for example by providing a upper bound of solution evaluations. Metaheuristics often generate high-quality solutions for difficult optimization problems.

Metaheuristics perform an incomplete search in the space of solutions by iteratively creating and evaluating new candidate solutions. While there exists a wide range of metaheuristic approaches, all use the same concepts of intensification (exploitation) and diversification (exploration) (Blum and Roli, 2003). During intensification steps, the algorithms try to improve the quality of solutions. In contrast, in diversification steps new areas are explored, which can lead to temporarily lower-quality solutions but helps the overall search process.

Basic and common design elements of metaheuristics are the representation, the search operators, the fitness function, the initialization and the search strategy (Rothlauf, 2011). Representations map candidate solutions (phenotypes) to a usually linear data structure (genotype). Search operators like crossover and mutation modify solutions and derive new solutions that are evaluated using the fitness function. The initialization is responsible for creating an initial set of solutions, from which the optimization process starts. The search strategy controls the balance between the intensification and diversification phases.

One big drawback of metaheuristics is that the more problems a particular approach can solve, the lower the resulting average performance is. This trade-off between effectiveness and application range is to some extend explained by the no-free-lunch theorem by Wolpert and Macready (1997), which essentially states that black box optimization – an algorithm that does not use problem-specific information – is not better than random search when applied to a problem that is closed under permutation. Therefore, problem-specific assumptions are necessary for efficient problem solving. This is why applying a standard, non problem-specific approach often only works for small problem instances. In those cases, high-quality solutions can be found, but for larger real-world optimization problems these methods fail.

One approach to overcoming this limitation is the use of problem-specific adaptations (Bonissone et al., 2006; Droste and Wiesmann, 2003; Hauschild et al., 2012; Raidl and Julstrom, 2003a; Rothlauf, 2011). While this reduces the universality of the approach, the performance for the problem at hand can be increased. When designing optimization methods in a problem-specific way, we gather information about the structure of high-quality, or even bad, solutions. This knowledge is exploited by introducing a *bias* into the algorithm, which concentrates the search in areas with high-quality solutions or avoids regions with bad solutions. On the downside, using a bias may also mislead the search, since a wrong bias leads to low-quality

solutions. Also, a bias that is too strong is disadvantageous: it focuses the search only on specific areas and excludes solutions from the search.

A problem-specific bias can be considered in all key elements of a metaheuristic. Representations can incorporate heuristics that favor particular solutions or use a redundant encoding, which assigns a larger number of genotypes to high-quality phenotypes. Search operators can be biased to prefer high-quality solution features over low-quality ones when generating new solutions from existing ones. Then, some solutions are created with higher probability than others. On average this leads to better solutions compared to unbiased search operators. Using a-priori knowledge to bias initial solutions is done in a similar way. Instead of creating all solutions in the search space with similar probability, construction heuristics are used to create initial solutions which are expected to be better than random solutions. Metaheuristics are then able to start the search at promising regions of the search space, which usually leads to better results faster. Exploiting problem-specific knowledge in the fitness function is difficult because it already contains a bias, since the fitness functions is used to distinguish high-quality and low-quality solutions. An additional bias can be introduced to reward or penalize certain parts of a solution. Often this is done if infeasible solutions exist in the search space in order to guide the search towards regions with feasible solutions.

While problem-specific design of optimization methods is a way to overcome the limitations of metaheuristics, it is unclear (1) how the required problem knowledge can be generated and (2) how this knowledge can be used to systematically design efficient and effective problem-specific optimization methods.

1.1 Goal of this thesis

This thesis addresses those two issues using case studies on three different constrained spanning tree problems. This work (1) identifies and analyzes relevant structural properties of the spanning tree problems and (2) demonstrates how the search performance of (meta-) heuristics regarding solution quality and running time can be improved by considering problem-specific knowledge in the heuristic's different components.

To identify and analyze structural properties of spanning tree problems, the approach used in this thesis is to analyze what distinguishes high-quality solutions from low-quality solutions. Therefore, a thorough problem analysis is the first step to systematically design problem-specific metaheuristics. The approach needs four essential components: (a) an exact or heuristic approach to create high-quality or even optimal solutions for small to medium-sized problem instances, (b) the creation or selection of problem instances that are representative, but solvable, (c) a

method to create random and unbiased solutions, and (d) a list of relevant solution characteristics that will be analyzed. Approaches to creating high-quality solutions can usually be found in the literature and need to be implemented. They have to be fast enough to solve the selected problem instances in reasonable time. Creating uniformly random solutions is not an easy task, since the constraints of the problem have to be respected. The selection of relevant characteristics is problem-dependent. This thesis focuses on combinatorial optimization problems in graphs, where solutions are spanning trees. Relevant characteristics of spanning tree problems can be split into two categories: ones that describe properties of the edges like edge weight or edge orientation, and others that describe the structural properties of whole trees, like node degree distributions or the Wiener index. It is also useful to analyze the backbones – that is the tree without its leaf nodes – since the backbone has a large influence on the overall weight of a tree. This thesis studies three relevant and difficult constrained spanning tree based problems: the optimal communication spanning tree (OCST) problem, the quadratic minimum spanning tree (QMST) problem, and the bounded-diameter minimum spanning tree (BDMST) problem.

The search performance of metaheuristics can be increased by exploiting the gained insights to create efficient problem-specific metaheuristics. After analyzing properties of high-quality solutions, this is demonstrated by biasing the different components of metaheuristics. Since search operators have a high influence on performance, this work designs biased operators for the OCST and the QMST problem. These operators create solutions with properties similar to those of high-quality solutions with higher probability. For the OCST problem, this thesis also proposes an approach using a biased fitness function, which uses penalties to guide the search into promising regions of the search space. Problem-specific construction heuristics are proposed for all three problems and experiments analyze the influence on the overall search performance of metaheuristics that start the search from biased initial solutions. Experimental results with different biased components and on different problems confirm that, by introducing a bias the search performance of metaheuristics, measured by solution quality and running time, is increased. Hence, a proper consideration of problem-specific knowledge results in efficient and robust search performance. The choice of metaheuristic is not that important.

Overall, this thesis gives advice on how to generate problem-specific knowledge for constraint spanning tree problems and how to systematically exploit this knowledge by introducing an analogous bias in the different components of heuristic optimization methods.

1.2 Structure

This thesis by publication consists of five articles, that are either published or submitted to scientific journals. In detail the thesis is structured as follows:

Chapter 2 starts the work on the OCST problem. Since the problem is $\mathcal{NP}$-hard, many state-of-the art approaches are based on metaheuristics, especially evolutionary algorithms. Biased variants use search operators that prefer short edges since optimal solutions are similar to minimum spanning trees. In Section 2.2, we introduce the edge orientation as another relevant property of edges. An experimental study on a large set of test instances shows that high-quality solutions to Euclidean OCST instances contain disproportionately many edges that point towards the center of their tree. Section 2.3 then extends the existing crossover operators to incorporate that knowledge and adjusts the needed parameters. The final experiments (Section 2.4) compare the performance of evolutionary algorithms using the new operators to the existing approaches on OCST instances with different configurations. The results show that the best performance is achieved by considering edge weights and orientation when selecting building blocks (edges) in the crossover operators.

While Chapter 2 focuses on the properties of the edges contained in optimal solutions for the Euclidean OCST problem, Chapter 3 analyzes structural properties of whole trees. Section 3.2 compares the node degree distributions, the location of nodes with certain degrees and the Wiener index of high-quality solutions with low quality ones. The results provide new insights into the structure of high-quality OCST solutions. These results are used in Section 3.3 to design a simple tree construction heuristic that is able to create spanning trees that are similar to optimal ones. During tree construction, the structural properties identified in this chapter are used, as is the knowledge about edge orientation provided in Chapter 2. Our construction heuristic is compared to a state-of-the art heuristic (Section 3.4) and the average solution quality is similar. In the final experiments, we use the construction heuristic to bias the initial population of an evolutionary algorithm. The results show the merits of using a biased initialization: the algorithm converges faster and better solutions are found.

For the OCST problem, different ways of including problem-specific knowledge are known. In the previous chapters, we showed how biasing the search operators (Chapter 2) and biasing the initial solutions (Chapter 3) increases search performance. The link-biased encoding (Rothlauf, 2009a) is a redundant representation that creates a bias towards low-weight edges by over-representing the solutions with desired properties. Chapter 4 presents an approach for the OCST problem that modifies the fitness function to create a bias towards low-weight edges that point towards the center of the graph. We use an approach called *guided local search* (GLS) that assigns costs to solution features – in our case, the graph's edges (Section 4.3).

A local search is performed until a local optimum is reached, then solution features that create the highest costs are penalized. In contrast to other approaches that favor certain types of edges, the GLS approach causes the search to avoid regions of the search space. Results (Section 4.4) show that such an approach based on penalties gives a robust search performance, independent of the type of problem-specific knowledge used.

Chapter 5 addresses the QMST problem, where the costs of a solution tree are the sum of the edge weights and the so-called intercosts that arise for every pair of edges in the solution. Since costs are linked additively, the problems looks significantly different depending on the edge weights and intercosts, respectively. This fact was neglected by previous work on the QMST problem. The problem also gets considerably easier if the intercosts do not play a large role. In this chapter we develop metaheuristic approaches that work independently of the ratio between edge weights and intercosts. First, in Section 5.3 we develop construction heuristics that favor edges with desired properties. The knowledge thus gained is used to create biased mutation and crossover operators (Sections 5.4 and 5.5). Using these search operators in two metaheuristics, an evolutionary and a simulated annealing algorithm, shows the superiority of these search operators compared to existing, unbiased ones. Finally, results in Section 5.6 show that we are able to successfully outperform the current state-of-the-art approaches for the QMST problem.

Chapter 6 considers construction heuristics for the BDMST problem. In a first step, the shortcomings of the existing approaches are analyzed by studying the structure of the backbones they create (Section 6.3). The backbone, that is the tree without its leaf nodes, has a huge influence on the overall costs for BDMST problems. A lightweight backbone that spans large parts of the graph, allows the leaf nodes to be connected with short edges. Otherwise, long edges are needed to connect the remaining nodes, which leads to poor solution quality. Based on the results of the analysis, two new heuristics are proposed (Section 6.4) and analyzed. Results show that they are able to create spanning trees that are similar to optimal ones. The final experiments (Section 6.5) analyze the performance of the new approaches for medium to large BDMST instances with different diameter bounds. Especially for large instances with tight diameter bounds, we are able to significantly improve the search performance compared to that of existing heuristics.

Finally, Chapter 7 summarizes and concludes the thesis.

Chapter 2

Edge orientation and the design of problem-specific crossover operators for the OCST problem

Wolfgang Steitz, Franz Rothlauf

Abstract

In the Euclidean optimal communication spanning tree (OCST) problem, the edges in optimal trees not only have small weights but also point with high probability towards the center of the graph. These characteristics of optimal solutions can be used for the design of problem-specific evolutionary algorithms (EAs). Recombination operators of direct encodings like edge-set and NetDir can be extended such that they prefer not only edges with small distance weights but also edges that point towards the center of the graph. Experimental results show higher performance and robustness in comparison to EAs using existing crossover strategies.

2.1 Introduction

The optimal communication spanning tree (OCST) problem (Hu, 1974) is an $\mathcal{NP}$-hard combinatorial optimization problem which seeks a spanning tree that satisfies all communication requirements at minimum total cost. Researchers have studied various approaches to solving the problem (Rothlauf, 2006; Sharma, 2006). Current state-of-the-art approaches are based on heuristics and metaheuristics, in particular evolutionary algorithms (EA).

The edge-set encoding (Raidl and Julstrom, 2003a) is a direct representation for trees, that encodes trees as sets of edges and uses encoding-specific search operators to generate candidate solutions. Search operators for edge-sets are either heuristic and rely on edge weights, or are non-heuristic. Mutation as well as crossover operators are based on a randomized version of Kruskal's algorithm. Another direct representation for the OCST problem is the NetDir encoding (Rothlauf, 2006). The crossover operator of this encoding copies subtrees from parents to offspring. In the original specification, no heuristics are incorporated into these operators.

High-quality solutions to Euclidean OCST problem instances contain disproportionately many edges that point towards the centers of their trees. This observation is used to systematically design problem-specific optimization methods. In EAs for the OCST problem that encode candidate trees using the edge-set and NetDir encodings, effective recombination operators can consider both edges' weights and orientations in constructing offspring trees. Experimental results reveal improved EA performance.

The main findings of this work are:

1. Edges pointing toward the center of the graph are overrepresented in good solutions to Euclidean OCST problem instances.

2. A greedy heuristic search performs best when edge insertion is controlled by a weighted combination of distance weights and edge orientation. Search performance can be further improved if orientation is neglected for edges next to the center of the graph.

3. EAs using direct encodings like edge-set or NetDir encoding show the highest performance if both edge properties, weight and orientation, are considered for edge-selection in the recombination operator.

The following paragraphs define the OCST problem, describe how to determine optimal solutions for small problem instances, and present properties of optimal solutions. Section 2.2.3 examines edge orientation in optimal solutions. Section 2.3 develops recombination operators that consider edge weights and orientation for

edge-set and NetDir encoding. Section 2.4 studies EA performance for different types of test instances.

2.2 The OCST problem

2.2.1 Problem definition

The optimal communication spanning tree (OCST) problem is a combinatorial tree optimization problem, first described by Hu (1974). Given a collection of nodes and the distances and communications demands between them, we seek a tree that connects all the nodes at minimum total communications cost, defined to be the sum over all pairs of nodes of the products of the distances and communications demands between them.

Formally, let $G = (V, E)$ be a complete, weighted, undirected graph with $n = |V|$ nodes and $m = |E|$ edges. Communication or transport requirements are given a priori in an $n \times n$ *demand matrix* $R = (r_{ij})$. Analogously, an $n \times n$ *distance weight matrix* $W = (w_{ij})$ specifies distance weights. In the Euclidean case, the distance weights are the Euclidean distances between the nodes. The weight $w(T)$ of a tree $T = (V, F)$ with $F \subseteq E$ and $|F| = n - 1$ is

$$w(T) = \sum_{i,j \in V} r_{ij} \cdot pl_{ij}, \tag{2.1}$$

where pl_{ij} denotes the path length between nodes i and j, which is calculated as the sum of the weights of all edges on the path between i and j. It depends on the structure of T and the distance matrix W. For spanning trees there is one unique path between any pair of nodes, and the path length pl can be determined using a depth first search. T is the optimal communication spanning tree if $w(T) \leq w(T')$ for all other spanning trees $w(T')$.

The OCST problem is $\mathcal{NP}$-hard (Garey and Johnson, 1979, p. 207). Furthermore, Reshef (1999) showed that the problem is $\mathcal{MAX}$ $\mathcal{SNP}$-hard. Therefore, no polynomial-time approximation scheme exists, unless $\mathcal{NP} = \mathcal{P}$ (C. Papadimitriou and Yannakakis, 1988). Exact polynomial-time algorithms exist for simplified variants of this problem where either the distance weights or the demands are uniform (Gomory and Hu, 1961; Hu, 1974; Wu and K.-M. Chao, 2004). For the general OCST problem, various approximation algorithms have been developed (Peleg and Reshef, 1998; Wu, K.-M. Chao, and Tang, 2000a; Wu, K.-M. Chao, and Tang, 2000b); however, due to the $\mathcal{MAX}$ $\mathcal{SNP}$-hardness of the problem the solution quality of such approximation algorithms is limited. Many heuristics, especially EAs, have been developed (Fischer and Merz, 2007; Y. Li and Bouchebaba, 2000; C. C. Palmer,

1994; Raidl and Julstrom, 2003a; Rothlauf, 2006; Rothlauf, Goldberg, and Heinzl, 2002; Soak, 2006).

The distance between two spanning trees T_i and T_j is half the number of their edges that are not common to both trees. Thus, the distance $d_{ij} \in \{0, 1, \ldots, n-1\}$ can be calculated as

$$d_{ij} = \frac{1}{2} \sum_{u,v \in V, u<v} |e^i_{uv} - e^j_{uv}|, \tag{2.2}$$

where $e^i_{uv} = 1$ if edge e_{uv} is included in T_i and $e^i_{uv} = 0$ otherwise.

2.2.2 Experimental design

The present studies follow Raidl and Julstrom (2003a) and use randomly created OCST test instances. The real-valued distance weights w_{ij} are the Euclidean distances between nodes i and j which are randomly placed on a 2-dimensional 10×10 grid.

Two types of demand are used. First, real-valued demands r_{ij} are randomly created and uniformly distributed in $]0, 10]$. Secondly, random demands following the Zipf distribution (Zipf, 1949) are created. The Zipf distribution is a power law probability distribution, which characterizes many natural phenomena like Internet traffic, population distribution and word usage (Li, 1992; Poosala, 1995). A discrete Zipf distribution is described by:

$$P_z(x) = \frac{1}{x^z} \cdot \frac{1}{\sum_{i=1}^{N} \frac{1}{i^z}}, \tag{2.3}$$

where P_z denotes the probability of $x \in \{1, 2, \ldots, N\}$. Zipf distributed demands are used with $z = 1$ and $N = 10$.

For finding optimal, or at least near-optimal solutions of the OCST problem, a mathematical programming solver is used for small problem instances with $n \leq 12$ and a genetic algorithm (GA) for larger problem instances. The OCST problem is modeled as an integer linear program (Rothlauf, 2007), and CPLEX 10.2 is able to solve all problem instances with $n \leq 12$ in a reasonable time.

The situation is different for larger problem instances ($n > 12$), which cannot be solved by CPLEX in reasonable time, and so an iterative GA is used. The GA is designed in such a way that it can be assumed that the solution found is optimal or near-optimal. First, a standard GA is applied n_{iter} times to an OCST problem instance using a population size of N_0. T_0^{best} denotes the best solution found during the n_{iter} runs. In the second round, a GA is again applied n_{iter} times, with $N_1 = 2N_0$, which returns the best solution T_1^{best}. The iterations continue and the population size $N_i = 2N_{i-1}$ is doubled, until $T_i^{best} = T_{i-1}^{best}$ and $n(T_i^{best})/n_{iter} > 0.5$; this means

that T_i^{best} is found in more than 50% of the runs in round i. $n(T_i^{best})$ denotes the number of runs that find the best solution T_i^{best} in round i.

For the experiments, a standard, generational GA with crossover and mutation is used. For the encoding of trees, Network Random Keys (NetKeys) (Rothlauf, Goldberg, and Heinzl, 2002) are used. NetKeys represent a tree using a list of continuous weights, one for each edge. The weights define an order of the edges, from which the tree is constructed using Kruskal's minimum spanning tree algorithm. Any string of weights is a valid chromosome and standard positional crossover and mutation operators may be used. NetKeys have high locality (Raidl and Gottlieb, 2005; Rothlauf and Goldberg, 1999), which leads to high EA performance (Rothlauf, 2006), and are unbiased, which means that the encoding does not favor a specific type of tree, but the probability of finding an optimal solution does not depend on its structure. The GA uses uniform crossover and tournament selection without replacement. The tournament size is three. Crossover probability is set to $p_c = 0.7$ and mutation probability (assigning a random value $[0, 1]$ to one allele) is set to $p_m = 1/l$, where $l = n(n-1)/2$. The effort required to find optimal or near-optimal solutions is high.

Uniformly random trees are created via Prüfer numbers (Prüfer, 1918) using Bijection between spanning trees and non-negative integers (Even, 1973, pp. 103-104). Using random Prüfer numbers yields unbiased random spanning trees. In contrast, randomized versions of Kruskal's and Prim's algorithms favor star-like trees (Julstrom and Raidl, 2002).

2.2.3 Properties of optimal solutions

Optimal solutions for OCST instances are biased towards minimum spanning trees (MSTs) (Rothlauf, 2009a). Therefore, average distances between optimal solutions and MSTs are significantly smaller than distances between optimal solutions and random trees. The performance of heuristic optimization methods can be increased by biasing search operators towards MST-like solutions. Edge-sets using heuristic search operators (Raidl, 2000; Raidl and Julstrom, 2003a) make use of this fact by preferring edges with small distance weights (Rothlauf, 2009b).

The following paragraphs report a study of the orientation of edges in optimal solutions to identify additional properties of high-quality solutions. Results show that edges directed towards the center of the graph are overrepresented in optimal solutions. Such edges lead to shorter paths and hence to a lower cost solution.

How can this observation be explained? Kershenbaum (Kershenbaum, 1993) distinguished between leaf and interior nodes and observed that it is useful to run more traffic over nodes near the center of a graph than over nodes far away from the center. Leaf nodes have only one neighbor and all traffic arriving at that node

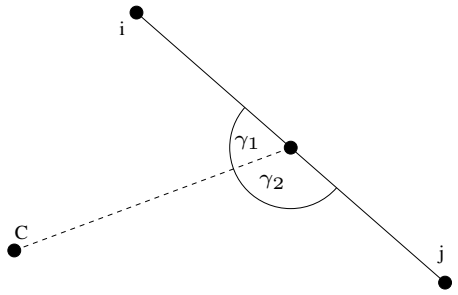


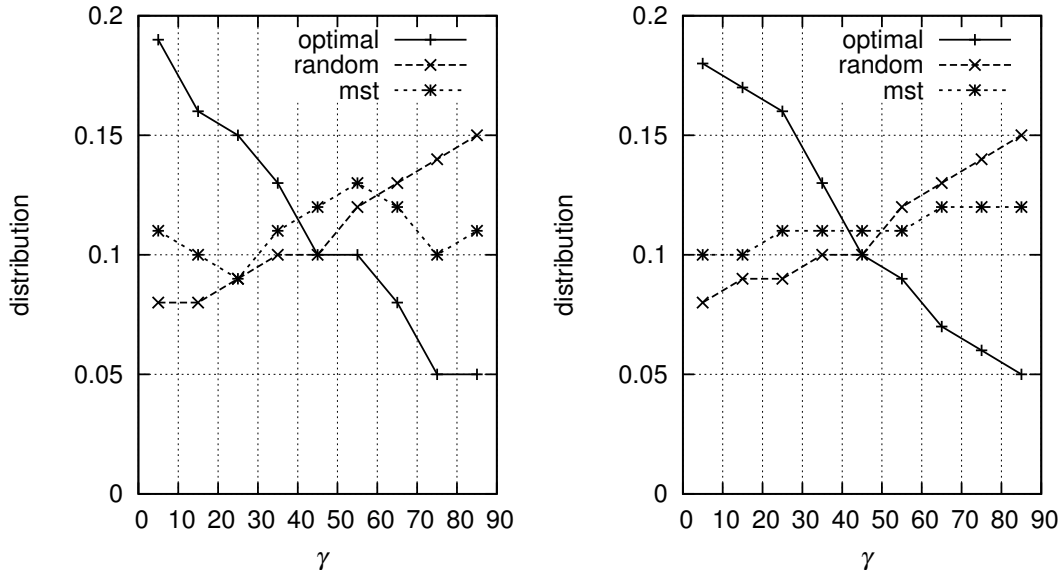
Figure 2.1: Orientation of an edge e_{ij}

also terminates there. In contrast, at interior nodes only some traffic terminates, and the remainder transits the node. Consequently, edges near the center of the graph mostly carry transit traffic whereas edges far away from the center carry almost no transit traffic. A tree construction method favoring edges pointing towards the center of the graph yields trees of lower cost, since such edges are the shortest connection from leaf nodes to interior nodes. Traffic originating from leaf nodes is directly routed to the center and from there to other parts of the tree. Therefore, it is more promising to connect leaf nodes to interior nodes using edges that point towards the center instead of connecting leaf nodes with other leaf nodes using tangential edges. In addition, it makes no sense to consider the orientation of edges that are close to the center of a graph, since the orientation of edges connecting interior nodes is meaningless.

Figure 2.1 illustrates edge orientation. The center C of the tree is calculated as the average of the x-coordinates and y-coordinates of all nodes. The orientation of an edge e_{ij} is the angle $\gamma \in [0, 90]$ between e_{ij} and the line connecting the midpoint of e_{ij} and the center C of the tree. There are two angles γ_1 and γ_2 with $\gamma_1 + \gamma_2 = 180$. The edge orientation is defined as the smaller angle $\gamma = min(\gamma_1, \gamma_2)$. $\gamma = 0$ for edges pointing directly towards the center.

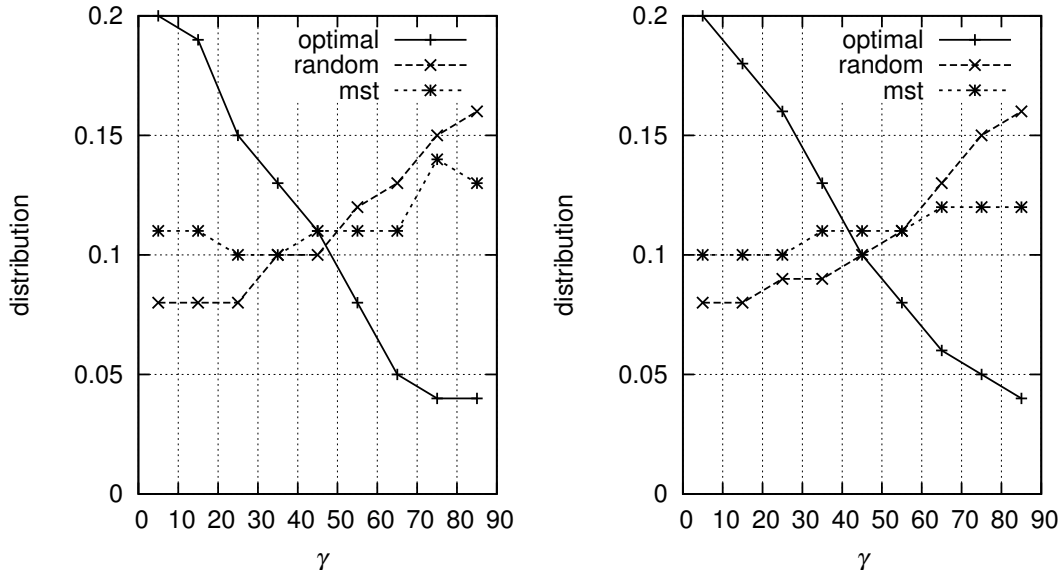
Experiments are performed to compare edge orientation for optimal solutions, randomly created spanning trees, and MSTs. For each problem size, 1,000 random Euclidean OCST test instances are created (compare Section 2.2.2). For each OCST instance, 10,000 random trees are generated, the MST is calculated, and an optimal (or near-optimal) solution is determined as described in Section 2.2.2.

Figure 2.2 presents results for $n = 15$ (Figs. 2.2a and 2.2b) and $n = 20$ (Figs. 2.2c and 2.2d). The demands are either uniformly distributed (Figs. 2.2a and 2.2c) or Zipf distributed (Figs. 2.2b and 2.2d). The figures plot histograms of γ for optimal solutions ("optimal"), random solutions ("random"), and MSTs ("mst"). Angles between $0 - 10, 10 - 20, 20 - 30, \ldots$ are considered identical. If the distribution is approximately uniform, orientation does not matter and all angles γ occur with



(a) $n = 15$, uniform demands

(b) $n = 15$, Zipf demands



(c) $n = 20$, uniform demands

(d) $n = 20$, Zipf demands

Figure 2.2: Distribution of orientation γ for optimal solutions, random solutions, and MSTs for 1000 randomly generated OCST instances with $n = 15$ and $n = 20$ nodes and different demand distributions

approximately the same probability in a tree. For random trees, edges with larger γ are slightly preferred. For MSTs, edge orientation is of less importance and γ is approximately uniformly distributed. For optimal solutions, γ is non-uniformly distributed, since edges with small γ occur more often. For $n = 20$, approximately 20% of all edges of optimal solutions have edge orientation with $\gamma \leq 10$, whereas only approximately 4% of all edges have $\gamma > 80$. These results support the hypothesis that edges pointing towards the center of a graph are preferred in optimal solutions.

2.3 Problem-specific EAs for the OCST problem

Optimal solutions have a bias towards edges with small γ for both demand distributions types, uniform and Zipf. This study investigates whether heuristic optimization methods can make use of this observation by favoring edges with small distance weight *and* small edge orientation γ.

2.3.1 Direct representations for trees

Edge-set

The edge-set (ES) encoding (Raidl, 2000; Raidl and Julstrom, 2003a) is a direct representation which encodes trees as sets of edges. ES operators are either heuristic and rely on edge weights, or are non-heuristic. Heuristic crossover operators result in higher performance in comparison to non-heuristic operators (Raidl and Julstrom, 2003a; Rothlauf, 2009b).

Crossover creates an offspring from two parental trees $T_1 = (V, E_1)$ and $T_2 = (V, E_2)$ by iteratively selecting edges from $F = (E_1 \cup E_2)$. Therefore, offspring trees consist solely of parental edges. Crossover operators differ according to the strategy used for selecting and inserting parental edges into offspring. Julstrom and Raidl (2001) studied several edge-selection strategies: tournament, greedy, inverse weight proportional, and random edge-selection. Results indicate that edge-selection using tournaments leads to the highest and most robust EA performance (Julstrom and Raidl, 2001; Steitz and Rothlauf, 2008). n-tournament edge-selection iteratively selects n edges, compares the associated edge weights, and inserts the edge with smallest weight into the offspring. This strategy has a bias towards low-weighted edges and MSTs (Raidl and Julstrom, 2003a; Rothlauf, 2009b). To strengthen the inheritance of common features from parents to offspring, edge-selection strategies can be designed as *-strategies (Raidl and Julstrom, 2003a). Then, all edges $(E_1 \cap E_2)$ are included in the offspring and remaining edges are selected from $F \setminus (E_1 \cap E_2)$ using an edge-selection strategy.

NetDir

The NetDir encoding (Rothlauf, 2006, Sect. 7.1) is a direct representation for trees, similar to ES. Originally, Rothlauf (2006) proposed only non-heuristic crossover and mutation operators. We extend this work and propose heuristic crossover operators which incorporate problem-specific knowledge.

NetDir crossover (Rothlauf, 2006) creates two offspring trees T_{o1} and T_{o2} from two parental trees $T_1 = (V, E_1)$ and $T_2 = (V, E_2)$ in two steps. The first step randomly assigns all nodes in V to one of the two disjoint sets V_1 and V_2. All edges E_1 of the first parent connecting nodes in V_1 (V_2) are inserted into offspring T_{o1} (T_{o2}). Analogously, all edges E_2 of the second parent connecting nodes in V_1 (V_2) are inserted into offspring T_{o2} (T_{o1}). The second step completes the two offspring trees by iteratively inserting randomly selected parental edges, which are not yet used in the offspring, until each offspring consists of $n - 1$ edges.

This crossover operator can easily be extended to include prior knowledge by modifying the second step. Instead of randomly selecting edges, we propose to use an edge-selection strategy that considers edge weights and edge orientation. The resulting heuristic crossover operator has similar properties to heuristic crossover operators for ES and is also biased towards MSTs.

Differences

ES and NetDir differ with respect to selection of edges that are transferred from parent to offspring. ES crossover does not transfer any subtrees to offspring but iteratively builds offspring from parental edges by adding single edges. In contrast, NetDir splits nodes into two sets and transfers all edges as well as subtrees that exist in one node set to an offspring. Therefore, NetDir crossover is, in principle, able to transfer meaningful subtrees from parent to offspring. However, the problem remains that there is no known algorithm of partitioning the nodes in such a way that meaningful subtrees are identified which can be transferred to an offspring; instead, nodes are randomly partitioned into two sets.

2.3.2 Extending crossover operators

The crossover operators are biased to consider knowledge about edge orientation. As a result, EAs are expected to find near-optimal solutions faster and more often. Thus, edges to be inserted into an offspring are not selected at random or according to their distance weights alone, but according to their weights *and* orientation. For this purpose, the modified weight w'_{ij} of an edge e_{ij} is introduced. It depends on the weight w_{ij} and orientation γ_{ij} of the edge and is defined as

$$w'_{ij} = \alpha w_{ij}/w_{max} + (1 - \alpha)\gamma_{ij}/\gamma_{max}, \tag{2.4}$$

where w_{ij} is the weight of e_{ij}, γ_{ij} denotes the orientation of e_{ij}, and $\alpha \in [0;1]$ is a parameter that controls the influence of w_{ij} and γ_{ij}. Distance weights as well as edge orientation are normalized using the maximum values $w_{max} = max(w_{ij})$ $(i,j = 1,\ldots,n)$ and $\gamma_{max} = max(\gamma_{ij})$ $(i,j = 1,\ldots,n)$. Therefore, $w'_{ij} \in [0,1]$.

Since edge orientation becomes meaningless if an edge is located near to the center of a graph, w''_{ij} is introduced. Edge orientation is considered only if the distance $d_{(ij),C}$ between an edge e_{ij} and the center C of a graph (see Figure 2.1) exceeds a predefined value. It is calculated as

$$w''_{ij} = \begin{cases} w'_{ij} & \text{if } dist_{(i,j),C}/dist_{max} \geq \beta \\ w_{ij}/w_{max} & \text{otherwise,} \end{cases} \tag{2.5}$$

where $dist_{max} = \max_{(i,j)} d_{(i,j),C}$.

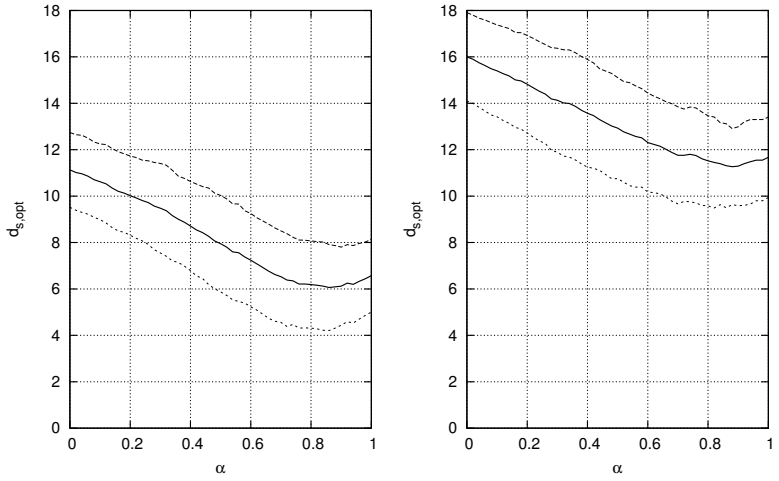
For ES as well as NetDir, edge-selection mechanisms can use w''_{ij} instead of w_{ij} for evaluating and selecting edges. With smaller w''_{ij}, the probability of an edge e_{ij} being included in an offspring increases. For $\alpha = 1$, only distance weights w_{ij} are considered, which is equal to the original ES crossover operator (see Section 2.3.1). For $\alpha < 1$, edge orientation influences the probability of edges being included in an offspring and edges pointing towards the center of a graph are preferred.

2.3.3 Balancing weight and orientation

This subsection neglects all crossover steps that do not consider edge weights (like transferring all common edges to an offspring in *-strategies) but focuses only on edge-selection strategies. It examines how performance of a greedy edge-selection strategy depends on α and β. Figures 2.3 and 2.4 present the best combination of α and β of the several values examined. The greedy edge-selection strategy starts with an empty tree and iteratively adds edges with minimum weight w''_{ij} until it is fully connected. Edges are taken from E; edges that would lead to cycles are omitted. For $\alpha = 1$, the greedy edge-selection strategy creates an MST.

The quality of a solution created by the greedy edge-selection strategy is determined by how closely it approaches an optimal solution. Results for 100 random OCST problem instances with 15 and 20 nodes are presented, respectively. Optimal solutions are determined according to Section 2.2.2.

Figure 2.3 shows the average distance $d_{s,opt}$ between trees T_s created by a greedy edge-selection strategy and optimal solutions T_{opt} versus α. The value of β is set to zero. Mean values are plotted as bold lines; standard deviations are plotted as regular lines. Results are consistent for different n. For $\alpha = 1$, an MST is created and, thus, $d_{s,opt}$ is the average distance between MSTs and optimal trees. Solution quality increases for $\alpha \approx 0.7 - 0.8$. Therefore, when both properties, orientation and distance weight are considered, the greedy edge-selection strategy can create



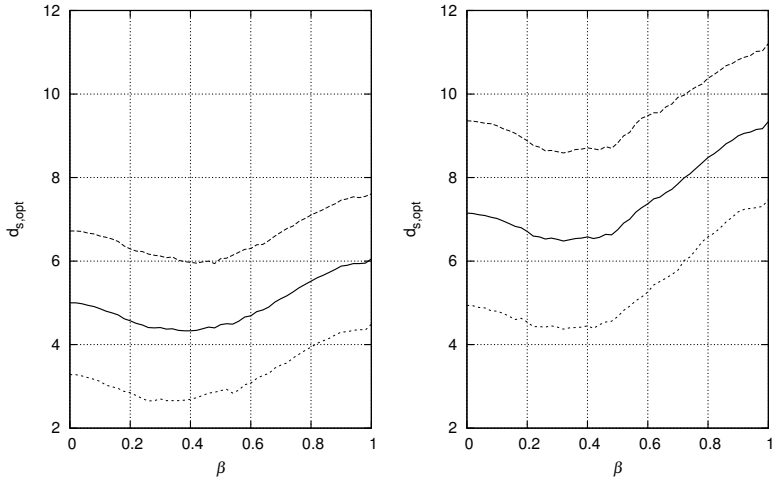
(a) $n = 15$ **(b)** $n = 20$

Figure 2.3: Average distance $d_{s,opt}$ between trees T_s generated by a greedy edge-selection strategy and optimal solutions T_{opt} versus α for randomly generated OCST instances with 15 and 20 nodes

better solutions. Finally, with smaller values of α, $d_{s,opt}$ increases. Considering only orientation ($\alpha = 0$) results in worse solutions than considering only distance weights ($\alpha = 1$).

Next, it is shown how the performance of a greedy edge-selection strategy using w'' depends on β. Figure 2.4 shows the average distance $d_{s,opt}$ versus β. The value of α is constant and set to $\alpha = 0.7$. Again, mean values are plotted as bold lines and standard deviations as regular lines. For $\beta = 1$, the greedy edge-selection strategy creates MSTs. For $\beta = 0$, the resulting $d_{s,opt}$ are equivalent to the smallest distances found in Figure 2.3. For $\beta \approx 0.3$, better solutions more closely approaching optimal trees can be obtained. Therefore, greedy edge-selection strategies create better solutions if orientation is considered only for edges which are some distance from the tree center.

Thus, it is recommended to consider distance weights as well as edge orientation, when constructing a tree by iteratively appending edges. Orientation should not be considered for edges near the tree center. In particular, we recommend setting $\alpha = 0.7$ and $\beta = 0.3$.



(a) $n = 15$ **(b)** $n = 20$

Figure 2.4: Average distance $d_{s,opt}$ between trees T_s generated by a greedy edge-selection strategy and optimal solutions T_{opt} versus β for randomly generated OCST instances with 15 and 20 nodes. The greedy algorithm subsequently inserts edges into a tree starting with edges of small w''_{ij}.

2.4 Performance of EAs using the extended operator

This section compares the performance of EAs considering distance weights and edge orientation in edge-set and NetDir. First, small problem instances where optimal (or near-optimal) solutions are determined as described in Section 2.2.2 are considered, then larger problem instances with unknown optimal solutions are analyzed. Following previous work (Raidl and Julstrom, 2003a; Rothlauf, 2009b), randomly created OCST test instances are used.

2.4.1 Small problem instances

A basic steady-state EA with non-heuristic initialization and mutation (Raidl and Julstrom, 2003a; Rothlauf, 2006) is used. The population size is 50. In each search step, either one (ES) or two (NetDir) offspring are created by crossover (crossover probability $p_c = 1$) and edge-wise mutation (mutation probability $p_m = 1/n$). The two parents are selected at random. If the cost of an offspring is smaller than or equal to the cost of the worst individual in the population ($w(T_{off}) \leq \max(w(T_i))$ for $i \in \{0, \ldots, N-1\}$), it replaces the worst individual in the population. This section

presents results for OCST instances of different size $n = \{10, 12, 14, 16, 18, 20\}$, where the optimal solutions are determined as described in Section 2.2.2. The demands are either uniformly distributed in $]0, 10]$ or Zipf distributed with $z = 1$ and $N = 10$. Each EA run terminates after $eval = 3000$ fitness evaluations. For every problem size, 100 Euclidean OCST instances are generated at random and for each problem instance 20 EA runs are performed.

Both ES and NetDir select edges using either binary tournaments with different settings of α and β or random edge-selection (denoted as RX), which results in an unbiased choice of offspring edges. The setting $\alpha = 1$, $\beta = 0$ is equivalent to using heuristic crossover and non-heuristic mutation (Raidl and Julstrom, 2003a; Rothlauf, 2009b).

Tables 2.1 and 2.2 list the percentage P_{suc} of runs that found solutions with the same objective value as T_{opt}, the average cost $w(T_{best})$ of the best solution found T_{best}, and the standard deviation σ of $w(T_{best})$ for the 100 instances. The table shows results for binary tournaments with different values of α and β, for random edge-selection, and for MSTs. The best results are printed in bold.

EA performance increases if heuristic edge-selection not only considers distance weights alone ($\alpha = 1$) but also edge orientation ($\alpha < 1$). By neglecting edges next to the center ($\beta = 0.3$), EA performance is further improved. The smallest average costs $w(T_{best})$ are observed for $\alpha = 0.7$ and $\beta = 0.3$. Results for ES and NetDir are consistent.

Previous results (Rothlauf, 2009b) indicate that the performance of heuristic ES is good if optimal solutions are similar to MSTs. Since optimal solutions for OCST instances are biased towards MSTs, ES performs well on many OCST problem instances. However, with increasing distance $d_{opt,mst}$ between optimal solutions and MSTs, EA performance drops sharply. Use of edge orientation as an additional edge-selection criterion is expected to improve the performance of heuristic ES for larger $d_{opt,mst}$. Figure 2.5 shows P_{suc} and the gap $\frac{w(T_{best})-w(T_{opt})}{w(T_{opt})}$ (percent) versus $d_{opt,mst}$ for 1000 random problem instances. Results are presented only for instances of size $n = 12$ and for uniformly distributed demand. Results for other problem sizes and Zipf demand distribution are analogous. For the 1000 problem instances, $\min(d_{opt,mst}) = 1$, $\max(d_{opt,mst}) = 7$.

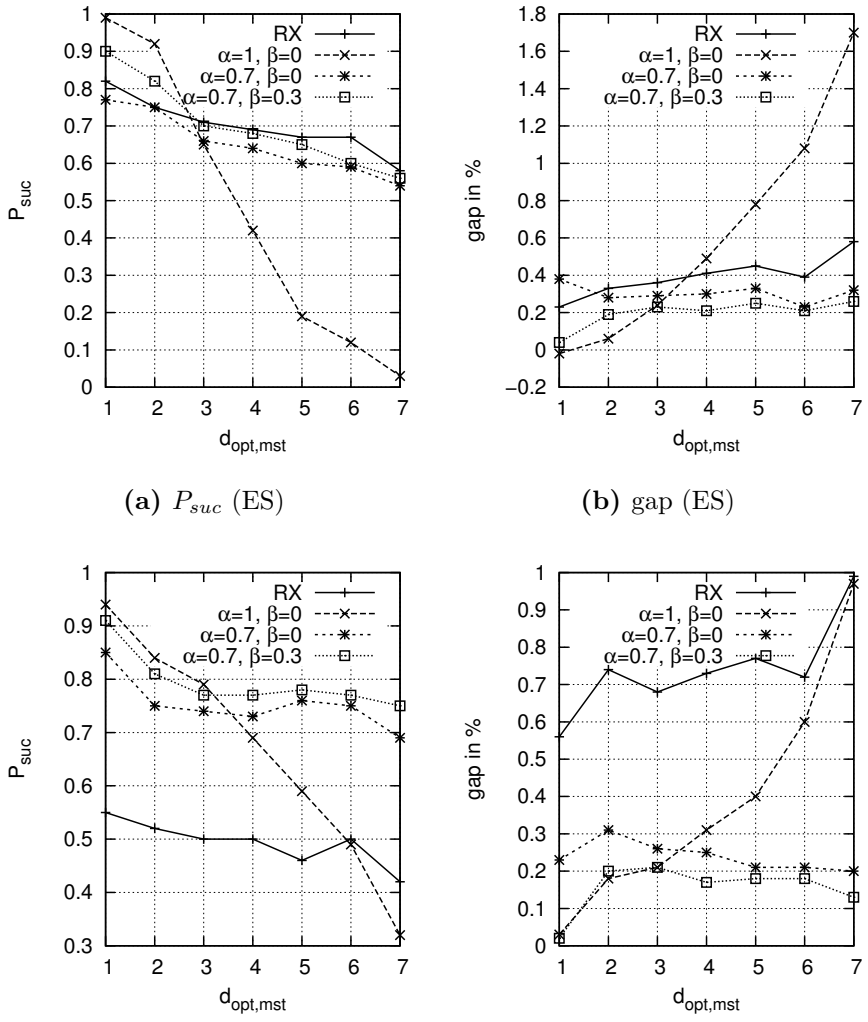
EA performance using edge-set with $\alpha = 1$ is high for OCST instances with small $d_{opt,mst}$. Since for these problem instances optimal solutions are only a few edges different from MSTs, edge-selection strategies that are based on distance weights alone are very successful. However, with increasing $d_{opt,mst}$, performance of EAs with $\alpha = 1$ drops sharply (Rothlauf, 2009b). In contrast, EAs using heuristic crossover which also consider edge orientation ($\alpha = 0.7$) show better performance for problems with larger $d_{opt,mst}$. By analogy with Figure 2.4, EA performance increases when neglecting edges next to the graph center ($\beta = 0.3$). Results for the

Table 2.1: EA performance for small problem instances with demand uniformly distributed in]0,10]

			Edge-Set								**NetDir**							
n		MST	RX	$\beta = 0$				$\beta = 0.3$			RX	$\beta = 0$				$\beta = 0.3$		
				α=1	α=0.8	α=0.7	α=0.6	α=0.8	α=0.7	α=0.6		α=1	α=0.8	α=0.7	α=0.6	α=0.8	α=0.7	α=0.6
10	P_{suc}	-	0.89	0.77	0.88	0.88	0.83	0.9	0.9	0.87	0.88	0.87	0.9	0.9	0.89	**0.91**	0.91	0.9
	$w(T_{best})$	1665.9	1489.9	1491.3	1490.0	1490.2	1491.0	1489.3	**1489.3**	1490.0	1490.1	1489.9	1489.6	1489.9	1490.0	1489.5	1489.4	1489.8
	σ	-	5	4.18	2.72	2.51	3.01	2.02	2.07	2.33	5.91	3.57	3.2	3.28	2.45	2.99	2.56	2.88
12	P_{suc}	-	0.69	0.4	0.64	0.63	0.56	0.66	0.68	0.62	0.49	0.65	0.76	0.75	0.69	0.77	**0.78**	0.75
	$w(T_{best})$	2519.1	2229.7	2234.3	2226.6	2227.2	2229.7	2225.8	2225.7	2227.5	2237.1	2228.9	2225.2	2225.9	2227.7	2224.9	**2224.6**	2225.6
	σ	-	17.19	11.03	6.79	6.89	7.98	5.91	5.45	6.56	25.41	12.65	8.57	8.82	9.88	8.14	7.44	8.06
14	P_{suc}	-	0.41	0.16	0.36	0.4	0.33	0.39	0.43	0.39	0.13	0.38	0.57	0.53	0.46	**0.6**	0.59	0.55
	$w(T_{best})$	3568.3	3132.0	3142.0	3123.6	3122.4	3128.2	3121.7	3119.9	3122.4	3167.7	3128.4	3118.7	3119.3	3123.8	**3116.3**	3116.8	3118.7
	σ	-	39.64	23.83	15.44	14.89	16.84	13.85	12.59	13.87	60.11	26.97	18.46	18.05	19.93	16.03	16.89	18.34
16	P_{suc}	-	0.11	0.04	0.14	0.14	0.11	0.18	0.21	0.19	0.01	0.12	0.23	0.24	0.15	**0.28**	0.28	0.25
	$w(T_{best})$	4839.3	4193.6	4191.4	4151.8	4155.6	4175.7	4147.2	**4143.6**	4152.3	4292.0	4175.8	4149.2	4154.1	4168.1	4146.3	4145.7	4150.4
	σ	-	76.44	41.41	27.63	30.9	36.86	27.1	24.12	24.3	121.53	57.2	38.72	42.17	47.59	41.72	39.24	39.25
18	P_{suc}	-	0.02	0.03	0.09	0.08	0.05	0.1	0.1	0.08	0	0.05	0.12	0.11	0.07	0.14	**0.14**	0.13
	$w(T_{best})$	6292.0	5505.6	5456.9	5376.9	5384.6	5423.5	5373.3	**5365.6**	5379.8	5709.9	5429.5	5384.2	5390.2	5421.8	5373.0	5370.2	5383.9
	σ	-	148.48	74.85	48.95	54.55	66.24	48.83	39.52	42.48	205	90.6	69.81	69.58	76.64	63.77	61.36	62.71
20	P_{suc}	-	0	0	0.02	0.03	0.01	0.04	0.05	0.03	0	0.01	0.03	0.04	0.01	0.04	0.04	0.04
	$w(T_{best})$	8006.8	7088.7	6934.7	6812.8	6817.0	6859.6	6802.8	**6784.1**	6794.0	7417.8	6940.1	6834.3	6830.6	6875.0	6829.8	6809.3	6812.0
	σ	-	237.66	102.45	74.59	74.73	90.45	69.18	58.36	52.72	315.24	152.83	114.76	110.27	117.46	112.65	93.06	95.82

Table 2.2: EA performance for small problem instances with demand Zipf distributed in [1,10]

			Edge-Set								**NetDir**							
n		MST	RX	$\beta = 0$				$\beta = 0.3$			RX	$\beta = 0$				$\beta = 0.3$		
				α=1	α=0.8	α=0.7	α=0.6	α=0.8	α=0.7	α=0.6		α=1	α=0.8	α=0.7	α=0.6	α=0.8	α=0.7	α=0.6
10	P_{suc}	-	0.87	0.77	0.82	0.83	0.8	0.83	0.81	0.79	0.81	0.85	0.87	0.88	0.84	**0.88**	0.86	0.84
	$w(T_{best})$	1142.7	1018.5	1018.4	1018.2	1018.4	1018.8	1018.0	1018.4	1019.2	1019.7	1018.0	1018.0	1017.9	1018.4	**1017.6**	1018.2	1018.2
	σ	-	3.73	3.1	2.12	2.21	2.01	2.05	2.11	2.15	5.94	2.95	2.81	2.82	3.1	2.2	3	2.19
12	P_{suc}	-	0.67	0.41	0.61	0.61	0.54	0.62	0.64	0.61	0.46	0.64	0.75	0.72	0.66	0.75	**0.75**	0.72
	$w(T_{best})$	1698.9	1495.6	1498.3	1493.2	1493.9	1496.1	1492.9	1493.0	1494.2	1500.5	1494.7	1492.3	1492.8	1494.4	1492.1	**1492.2**	1492.9
	σ	-	11.84	8.02	4.57	5.03	6.13	4.31	4.07	4.72	16.56	8.99	6.23	6.35	7.29	5.91	5.64	5.78
14	P_{suc}	-	0.39	0.18	0.4	0.43	0.35	0.42	0.46	0.39	0.13	0.38	**0.56**	0.54	0.46	0.55	0.55	0.53
	$w(T_{best})$	2458.8	2163.7	2167.4	2155.1	2157.0	2161.8	2154.2	2153.2	2156.6	2187.8	2160.3	2152.6	2153.4	2157.4	2152.5	**2152.1**	2154.1
	σ	-	28.01	16.91	9.97	11.72	14.35	9.94	9.07	10.1	44.53	21.56	12.95	12.97	15.09	13.6	12.51	13.76
16	P_{suc}	-	0.12	0.06	0.17	0.17	0.11	0.21	0.24	0.18	0.02	0.16	0.27	0.24	0.17	0.3	**0.32**	0.29
	$w(T_{best})$	3321.0	2821.7	2819.8	2793.8	2795.1	2809.7	2789.4	**2787.5**	2794.0	2881.4	2807.5	2792.6	2795.7	2807.5	2788.9	2788.8	2792.3
	σ	-	51.55	30.02	18.78	18.47	24.25	15.86	14.3	15.27	78.2	37.94	27.14	25.92	31.15	24.43	24.01	23.71
18	P_{suc}	-	0.02	0.01	0.07	0.08	0.07	0.08	0.11	0.09	0	0.03	0.11	0.11	0.1	0.11	**0.14**	0.1
	$w(T_{best})$	4270.7	3703.3	3671.6	3615.5	3618.9	3636.5	3612.8	**3608.1**	3613.7	3830.9	3662.6	3621.7	3619.9	3634.8	3617.7	3614.1	3619.0
	σ	-	95.42	50.18	29.09	30.84	37.33	29.11	26.46	27.38	137.44	66.21	49.9	45.02	49.11	46.36	44.8	41.22
20	P_{suc}	-	0	0	0.02	0.05	0.04	0.02	**0.06**	0.06	0	0.01	0.04	0.05	0.04	0.05	0.06	0.06
	$w(T_{best})$	5415.2	4677.5	4592.1	4494.1	4495.2	4528.3	4488.9	**4476.6**	4488.1	4890.5	4572.3	4510.8	4510.4	4537.7	4503.2	4494.0	4502.1
	σ	-	157.72	75.32	43.85	47.17	59.88	44.12	37.15	41.59	208.43	94.19	72.99	70.51	75.99	71.32	66.07	64.29



(a) P_{suc} (ES) **(b)** gap (ES)

(c) P_{suc} (NetDir) **(d)** gap (NetDir)

Figure 2.5: Edge set and NetDir performance using different edge-selection strategies for randomly generated OCST instances ($n = 12$). The tables shows average success probability P_{suc} versus $d_{opt,mst}$ (left) and average gap $\frac{w(T_{best})-w(T_{opt})}{w(T_{opt})}$ (percent) versus $d_{opt,mst}$ (right).

Table 2.3: Number of evaluations *eval* for larger problem instances

n	25	50	75	100
eval	5,000	20,000	40,000	80,000

gap $\frac{w(T_{best})-w(T_{opt})}{w(T_{opt})}$ are analogous to P_{suc}. In the figures, RX performs well in comparison to the more complex heuristic approaches. However, this is only the case for small n, since the gap increases with larger n (see Tables 2.4 and 2.5).

Overall, EAs using edge-selection strategies with $\alpha = 1$ perform well only if $d_{opt,mst}$ is small. With increasing $d_{opt,mst}$, edge-selection strategies that also consider edge orientation show better performance. Using edge-set or NetDir with heuristic edge-selection strategies ($\alpha = 0.7$, $\beta = 0.3$) results in a high and robust EA performance.

2.4.2 Larger problem instances

This section investigates larger OCST instances with unknown optimal solutions. EA performance is measured using $w(T_{best})$.

The same EA as in the previous experiments is used, but with a larger population size of 200. Since a larger number of evaluations improves EA performance, *eval* is increased with larger n (see Table 2.3). This section presents results for OCST instances with $n = \{25, 50, 75, 100\}$. For each problem size, 100 random instances are created. Due to computational restrictions, only 50 instances are considered for $n = 75$ and 25 for $n = 100$. For each problem instance, 20 independent EA runs are performed on a dual core Intel processor with 2 GHz and 4 GB RAM running 64bit Linux.

For different n, the tables list $w(T_{best})$ using either ES (Table 2.4) or NetDir (Table 2.5). Additionally, the standard deviations σ of $w(T_{best})$ and the average running time t_{cpu} (in seconds) are shown. EAs with heuristic crossover perform best if edge orientation is considered. EAs using ES or NetDir with $\alpha = 0.7$ and $\beta = 0.3$ outperform heuristic edge-selection strategies that consider only distance weights ($\alpha = 1$, $\beta = 0$). Differences are significant using a ranked t-test with an error level of $p < 0.01$. Also, differences between heuristic and non-heuristic (RX) variants are significant with an error level of $p < 0.0001$. Considering that edge orientation does not increase CPU times, however, NetDir needs twice as long in comparison to ES due to the more complicated crossover method.

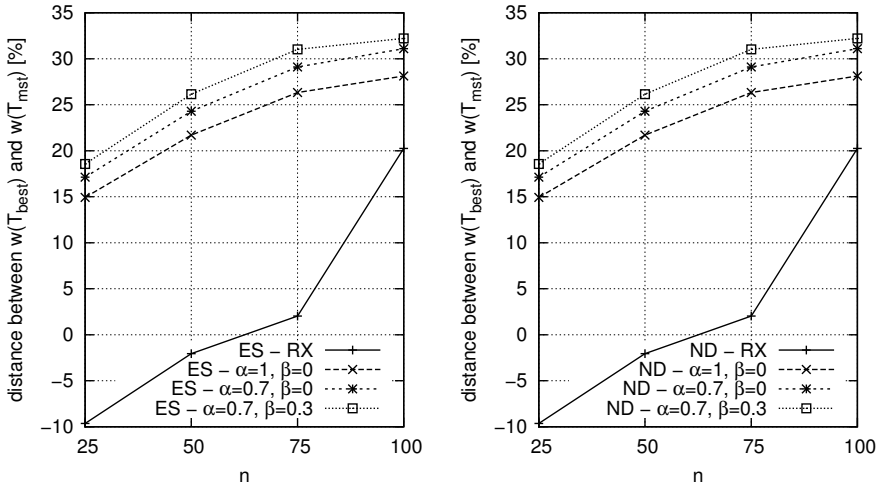
The plots in Figure 2.6 show the average gap $\frac{w(T_{mst})-w(T_{best})}{w(T_{mst})}$ (percent) between the best solution found, T_{best}, and an MST over n. A large gap indicates good EA performance. The MST has been chosen as reference since it is already a high-

Table 2.4: EA performance for large problem instances using ES

r_{ij}	n		MST	RX	$\beta = 0$				$\beta = 0.3$		
					α=1	α=0.8	α=0.7	α=0.6	α=0.8	α=0.7	α=0.6
uniform $]0, 10]$	25	$w(T_{best})$	13046.46	14336.4	11168.47	10792.95	10841.04	11090.93	10734.67	**10672.61**	10697.67
		σ	-	663.39	129.4	95.45	107.81	162.12	79.17	62.63	65.82
		t_{cpu}	-	0.46	0.46	0.45	0.45	0.45	0.45	0.45	0.44
	50	$w(T_{best})$	60414.6	61776.79	47217.46	45174.28	45703.93	46634.06	44661.38	**44470.87**	44530.74
		σ	-	5656.98	659.54	541.75	623.97	774.08	386.1	336.37	341.48
		t_{cpu}	-	7.85	7.98	7.5	7.32	7.18	7.5	7.33	7.2
	75	$w(T_{best})$	146666.49	144911.47	110239.47	104481.36	105705.18	107280.61	103620.07	**103179.69**	103203.24
		σ	-	19931.81	1783.95	1197.07	1441.39	1771.13	972.17	1053.69	1019.64
		t_{cpu}	-	39.42	40.92	37.3	36.37	35.71	37.4	36.45	36.06
	100	$w(T_{best})$	273907.56	217994.8	197470.94	188629.33	189753.68	191777.38	186734.87	**186601.03**	186616.35
		σ	-	18136.12	3590.61	2010.94	1989.15	2368.65	1426.39	1466.98	1272.11
		t_{cpu}	-	149.91	163.44	146.05	141.61	139.88	145.46	141.07	139.27
Zipf $[1, 10]$	25	$w(T_{best})$	8971.71	9836.46	7632.19	7381.63	7435.29	7602.21	7335.87	**7306.6**	7323.61
		σ	-	451.14	88.94	66.54	85.91	111.71	52.31	47.57	49.15
		t_{cpu}	-	0.46	0.46	0.45	0.45	0.45	0.45	0.45	0.44
	50	$w(T_{best})$	41360.71	42208.91	32385.65	30959.98	31310.29	31903.08	30696.32	**30546.99**	30631.24
		σ	-	3962.15	425.92	347.41	425.16	537.9	274.62	240.84	233.99
		t_{cpu}	-	7.85	7.93	7.47	7.28	7.15	7.47	7.31	7.19
	75	$w(T_{best})$	103171.16	98551.87	75860.84	71806.87	72440.23	73413.01	71206.76	70899.78	**70842.18**
		σ	-	13464.27	1274.22	771.67	998.17	1043.13	700.35	647.89	617
		t_{cpu}	-	39.38	41.23	37.44	36.4	35.75	37.41	36.43	35.88
	100	$w(T_{best})$	189905.54	147212.74	133607.99	126700.48	127705.89	129517.46	125752.76	**125296.04**	125380.23
		σ	-	12931.02	2373.63	1282.23	1457.66	1723.26	1102.86	1117.2	994.55
		t_{cpu}	-	149.75	161.65	142.18	137	133.49	144.13	139.47	137.56

Table 2.5: EA performance for large problem instances using NetDir

r_{ij}	n		MST	RX	$\beta = 0$				$\beta = 0.3$		
					α=1	α=0.8	α=0.7	α=0.6	α=0.8	α=0.7	α=0.6
uniform $]0, 10]$	25	$w(T_{best})$	13046.46	14227.2	11172.89	10855.55	10860.05	11019.85	10802.14	**10748.89**	10774.09
		σ	-	597.55	179.02	152.57	151.46	186.06	131.9	118.9	121.5
		t_{cpu}	-	0.61	0.61	0.6	0.6	0.6	0.6	0.6	0.6
	50	$w(T_{best})$	60414.6	63248.7	47190.02	45547.11	45863.24	46625.44	45197.72	**45003.69**	45101.67
		σ	-	3564.34	1057.19	859.94	918.09	1032.24	810.97	666.68	744.5
		t_{cpu}	-	14.6	14.05	13.96	13.97	13.96	13.97	14.02	13.97
	75	$w(T_{best})$	146666.49	164267.53	111914.36	107405.89	108694.24	111415.18	106300.68	**106120.8**	106470.56
		σ	-	10392.99	2847.32	2778.71	3042.12	4008.84	2333.4	2381.52	2524.53
		t_{cpu}	-	87.31	83.3	83.52	83.83	83.98	83.67	83.64	83.9
	100	$w(T_{best})$	273907.56	307756.16	201303.24	196095.84	200454.3	204455.03	192629.78	**192347.1**	194284.05
		σ	-	20344.12	6059.54	5962.87	7829.42	8481.6	4502.47	5068.37	5480.95
		t_{cpu}	-	391.48	375.04	374.31	374.43	376.82	373.5	373.75	375.12
Zipf $[1, 10]$	25	$w(T_{best})$	8971.71	9718.4	7637.61	7426.95	7449.66	7571.55	7383.1	**7356.23**	7374.92
		σ	-	395	133.76	106.4	109.18	122.75	88.7	83.76	86.91
		t_{cpu}	-	0.61	0.61	0.6	0.6	0.6	0.6	0.6	0.6
	50	$w(T_{best})$	41360.71	43544.69	32377.31	31175.54	31442.19	31941.43	30990.68	**30910.49**	31001.03
		σ	-	2534.16	697.63	589.28	602.04	669.04	514.42	480.12	499.17
		t_{cpu}	-	14.6	14.09	14.03	14.02	14.02	14.03	14.03	14.02
	75	$w(T_{best})$	103171.16	112989.81	77036.47	74244.53	74972.06	76455.63	73149.53	**73038.13**	73359.8
		σ	-	6652.43	2165.71	1980.75	2132.39	2282.15	1523.37	1542.47	1799.97
		t_{cpu}	-	87.16	83.22	83.11	83.58	83.56	83.17	83.38	83.57
	100	$w(T_{best})$	189905.54	208206.86	139200.24	134458.69	137368.24	140703.7	**129108.27**	129389.41	129783.93
		σ	-	16034.97	5182.14	4696.39	5848.25	7055.14	3268.43	3341.42	3174.71
		t_{cpu}	-	390.33	374.13	373.9	372.85	374.92	374.33	374.8	374.09



(a) Edge Set **(b)** NetDir

Figure 2.6: EA performance using different crossover variants for randomly generated OCST instances. The plots show the average gap between the cost of the best solution found and the MST over the problem size n. The larger the gap, the higher EA's performance.

quality solution for OCST problem instances. EAs with $\alpha = 0.7$, $\beta = 0.3$ perform best and are able to find high-quality solutions with larger distances from MSTs.

Overall, performance of EAs using heuristic crossover is good if edge orientation is considered when selecting the edges of offspring ($\alpha = 0.7$). Performance can be further improved if edges next to the center of the graphs are neglected ($\beta = 0.3$).

2.5 Summary and conclusions

This work studies the OCST problem and shows that edges in optimal solutions are not uniformly oriented, but edges pointing towards the center of the graph occur with higher probability. Thus, EA performance can be systematically improved by biasing the search operators to favor such edges.

This work exploits this property of optimal solutions for crossover operators of direct tree representations like edge-set and NetDir. In such representations, crossover creates offspring by iteratively selecting parental edges. An extended crossover operator is proposed which selects edges to be included in the offspring based on edge weights *and* edge orientation. Parental edges that have low weight and point towards the center of the graph are included with higher probability in an offspring.

Crossover operators using both criteria, weight and orientation, outperform existing approaches which consider only edge weights.

The results suggest using heuristic crossover operators which prefer edges that point towards the center of a graph and have small distance weights. Considering both criteria results in a robust EA performance, and problems where the optimal solution is quite different from MSTs can also be solved. While the focus in this work is on the OCST problem, the basic approach is generally applicable to other Euclidean graph problems.

Future work will address what other problem-specific knowledge of tree problems can be used for designing high-quality EAs. While current approaches use only properties of edges, future work will analyze properties of tree structures. Other promising areas are problem-specific mutation and initialization operators for edge-set or NetDir which consider edge orientation.

References

Even, S. (1973). *Algorithmic Combinatorics*. New York: The Macmillan Company.

Fischer, T. and P. Merz (2007). "A memetic algorithm for the optimum communication spanning tree problem". In: *Hybrid Metaheuristics*. Ed. by T. Bartz-Beielstein, M. J. B. Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels. Vol. 4771. Springer, pp. 170–184.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Gomory, R. E. and T. C. Hu (1961). "Multi-terminal network flows". In: *Journal of the Society for Industrial and Applied Mathematics* 9.4, pp. 551–570.

Hu, T. C. (1974). "Optimum communication spanning trees." In: *SIAM Journal on Computing* 3.3, pp. 188–195.

Julstrom, B. A. and G. R. Raidl (2001). "Weight-biased edge-crossover in evolutionary algorithms for two graph problems". In: *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing, Las Vegas, Nevada, United States*. New York, NY, USA: ACM, pp. 321–326.

Julstrom, B. A. and G. R. Raidl (2002). "Initialization is robust in evolutionary algorithms that encode spanning trees as sets of edges". In: *SAC '02: Proceedings of the 2002 ACM symposium on applied computing, Madrid, Spain*. New York, NY, USA: ACM, pp. 547–552.

Kershenbaum, A. (1993). *Telecommunications network design algorithms*. New York: McGraw Hill.

Li (1992). "Random texts exhibit Zipf's law-like word frequency distribution". In: *IEEE Transactions on Information Theory* 38.

Li, Y. and Y. Bouchebaba (2000). "A new genetic algorithm for the optimal communication spanning tree problem". In: *AE '99: Selected Papers from the 4th European Conference on Artificial Evolution*. London, UK: Springer-Verlag, pp. 162–173.

Palmer, C. C. (1994). "An approach to a problem in network design using genetic algorithms". PhD thesis. Troy, NY, USA: Polytechnic University.

Papadimitriou, C. and M. Yannakakis (1988). "Optimization, approximation, and complexity classes". In: *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing, Chicago, Illinois, United States*. New York, NY, USA: ACM Press, pp. 229–234.

Peleg, D. and E. Reshef (1998). "Deterministic polylog approximation for minimum communication spanning trees". In: *ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming*. London, UK: Springer-Verlag, pp. 670–681.

Poosala, V. (1995). *Zipf's Law*. Tech. rep. University of Wisconsin.

Prüfer, H. (1918). "Neuer Beweis eines Satzes über Permutationen". In: *Archiv für Mathematik und Physik* 27, pp. 742–744.

Raidl, G. R. (2000). "An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem". In: *Proc. of the 2000 Congress on Evolutionary Computation*. Piscataway, NJ: IEEE Service Center, pp. 104–111.

Raidl, G. R. and J. Gottlieb (2005). "Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: a case study for the multidimensional knapsack problem". In: *Evolutionary Computation* 13.4, pp. 441–475.

Raidl, G. R. and B. A. Julstrom (2003a). "Edge sets: an effective evolutionary coding of spanning trees". In: *IEEE Transactions on Evolutionary Computation* 7.3, pp. 225–239.

Reshef, E. (1999). "Approximating minimum communication cost spanning trees and related problems". MA thesis. Rehovot 76100, Israel: Feinberg Graduate School of the Weizmann Institute of Science.

Rothlauf, F. (2007). *Design and Application of Metaheuristics*. Habilitationsschrift. Department of Information Systems 1, University of Mannheim, Germany.

Rothlauf, F. and D. E. Goldberg (1999). "Tree network design with genetic algorithms - an investigation in the locality of the prüfernumber encoding". In: *Late Breaking Papers at the Genetic and Evolutionary Computation Conference 1999*. Ed. by S. Brave and A. S. Wu. Orlando, Florida, USA: Omni Press, pp. 238–244.

Rothlauf, F. (2006). *Representations for genetic and evolutionary algorithms (2. ed.)* Springer, pp. I–XVII, 1–325.

Rothlauf, F. (2009a). "An encoding in metaheuristics for the minimum communication spanning tree problem." In: *INFORMS Journal on Computing* 21.4, pp. 575–584.

Rothlauf, F. (2009b). "On Optimal Solutions for the Optimal Communication Spanning Tree Problem". In: *Operations Research* 57.2, pp. 413–425.

Rothlauf, F., D. E. Goldberg, and A. Heinzl (2002). "Network random keys: a tree representation scheme for genetic and evolutionary algorithms". In: *Evolutionary Computaton* 10.1, pp. 75–97.

Sharma, P. (2006). "Algorithms for the optimum communication spanning tree problem". In: *Annals of Operations Research* 143.1, pp. 203–209.

Soak, S.-M. (2006). "A new evolutionary approach for the optimal communication spanning tree problem". In: *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences* E89-A.10, pp. 2882–2893.

Steitz, W. and F. Rothlauf (2008). "Orientation matters: how to efficiently solve OCST problems with problem-specific EAs". In: *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, Atlanta, GA, USA*. New York, NY, USA: ACM, pp. 563–570.

Wu, B. Y. and K.-M. Chao (2004). *Spanning Trees and Optimization Problems*. CRC Press.

Wu, B. Y., K.-M. Chao, and C. Y. Tang (2000a). "A polynomial time approximation scheme for optimal product-requirement communication spanning trees". In: *Journal of Algorithms* 36.2, pp. 182–204.

Wu, B. Y., K.-M. Chao, and C. Y. Tang (2000b). "Approximation algorithms for some optimum communication spanning tree problems". In: *Discrete Applied Mathematics* 102.3, pp. 245–266.

Zipf, G. K. (1949). *Human Behaviour and the Principle of Least Effort: an Introduction to Human Ecology*. Addison-Wesley.

Chapter 3

New insights into the OCST problem: integrating node degrees and their location in the graph

Wolfgang Steitz, Franz Rothlauf

Abstract

This paper considers the Euclidean variant of the optimal communciation spanning tree (OCST) problem. Researchers have analyzed the structure of the problem and found that high quality solutions prefer edges of low cost. Further, edges pointing to the center of the network are more likely to be included in good solutions. We add to the literature and provide additional insights into the structure of the OCST problem. Therefore, we investigate properies of the whole tree, such as node degrees and the Wiener index. The results reveal that optimal solutions are structured in a star-like manner. There are few nodes with high node degrees, these nodes are located next to the graph's center. The majority of the nodes have very low node degrees. Especially, nodes with degree one are very common and located far away from the center. We exploit these insights to develop a construction heuristic, which builds spanning trees with similar properties. Experiments indicate a high solution quality for the OCST problem. In a next step, we seed the initial population of an evolutionary algorithm (EA) with solutions constructed with our method. An experimental study demonstrates the merits of using a biased initialization: the algorithm is faster and better compared to the same algorithm using random starting solutions.

3.1 Introduction

The optimal communication spanning tree (OCST) problem (Hu, 1974) is a common $\mathcal{NP}$-hard combinatorial optimization problem which seeks a spanning tree that satisfies all communication requirements and leads to minimal total costs. Researches studied various solution approaches for the OCST problem (Rothlauf, 2006; Sharma, 2006). The current state-of-the-art approaches are based on heuristics and metaheuristics, in particular evolutionary algorithms (EA).

One can improve the performance of EAs by incorporating problem-specific knowledge into the search operators, which guide the search to promising solutions more quickly and better solutions are found faster. One option to systematically use problem-specific knowledge in the optimization method, is to start with analyzing the properties of high-quality solutions. If there are any significant differences between high-quality and random solutions, designing optimization methods to exploit this knowledge leads to high-performing optimization methods. Including problem-specific knowledge in heuristic optimization methods can be done in various different ways, like developing new search operators, changing the fitness function or changing the used representation. One method, called inoculation (Surry and Radcliffe, 1996), is to seed the initial population with solutions found by heuristic initialization methods. Using this method, the primary optimization method starts from good areas in the search space and therefore quickly reaches high-quality solutions.

In this paper we analyze the properties of the Euclidean variant of the OCST problem. It is already known that low-weight edges are preferred in high-quality solutions (Rothlauf, 2009a). Also, edges pointing to the graph's center are more likely to be included in good solutions (Steitz and Rothlauf, 2008). Our analysis investigates additional properties, which regard the structure of the whole tree, such as node degrees and the Wiener index. The results reveal that optimal solutions are structured in a star-like manner. There are few nodes with high node degrees and these nodes are located next to the graph's center. The majority of the nodes have very low node degrees. Especially, nodes with degree one are very common and located far away from the center. In a next step, we present a heuristic and straight forward initialization method for the OCST problem. This approach creates spanning trees with similar properties as high-quality solutions. We compare this approach to the best known construction heuristic for the OCST problem (Ahuja-Murty Tree Building) (Ahuja and Murty, 1987a), which is complex and based on approximation of the total costs. Our approach creates spanning trees with similar quality, using less computation time. In a second step, we analyze the performance of a simple steady-state EA using different initialization methods. Results of these experiments show that we reach better solutions using inoculation, but there is no significant difference between the two methods.

The main results presented in this paper are:

1. High-quality solutions for the Euclidean variant of the OCST problem have on average a low Wiener index and are therefore very star-like.

2. In optimal solutions, there are always a few nodes with very high node degrees. These hubs are located next to the graph's center.

3. More than half of the nodes are leaf-nodes (node degree one) and located far away from the center.

4. A straight forward heuristic tree construction method using the gained knowledge creates spanning trees with similar properties. These solutions have similar fitness compared to Ahuja-Murty Tree Building, the state-of-the-art approach found in the literature.

5. Seeding the initial population of an EA with spanning trees created by one of the construction heuristics improves EA-performance.

The following section defines the OCST problem, presents properties of optimal solutions and describes how to determine optimal solutions for small problem instances of the OCST problem. Additionally, it provides statistical analysis of various graph properties of high-quality solutions. In Sec. 3.3, two different initialization methods are presented. In Sec. 3.4, we present experimental results of the performance of these two initialization methods. The paper ends with concluding remarks.

3.2 The OCST problem

3.2.1 Definition

The OCST problem is a common $\mathcal{NP}$-hard combinatorial optimization problem and was first introduced by Hu (1974). The goal is to determine a tree which connects all given nodes and satisfies their communication requirements for a minimum total costs. It can be formulated as follows: Let $G = (V, E)$ be a weighted, undirected graph with $n = |V|$ nodes and $m = |E|$ edges. The communication or transport requirements between the n different nodes are given a priori in the $n \times n$ *demand matrix* $R = (r_{ij})$. Analogically, the $n \times n$ *distance weight matrix* $W = (w_{ij})$ specifies the distance weights. The weight $w(T)$ of a tree $T = (V, F)$ with $(F \subseteq E)$ and $|F| = n - 1$ is calculated as follows:

$$w(T) = \sum_{i,j \in V} w_{ij} \cdot b_{ij}, \tag{3.1}$$

where b_{ij} denotes the traffic flowing directly or indirectly over the edge between nodes i and j.

$$b_{ij} = \sum_{i \in V_{ij}, j \in W_{ij}} (r_{ij} + r_{ji}), \tag{3.2}$$

where V_{ij}, W_{ij} are the components of T when edge (i, j) is removed. The traffic is calculated according to the structure of T and the demand matrix R. T is the optimal communication spanning tree, if $w(T) \leq w(T')$ for all other spanning trees $w(T')$.

To measure the difference between two spanning trees T_i and T_j, the distance $d_{ij} \in \{0, 1, \ldots, n-1\}$ can be calculated as

$$d_{ij} = \frac{1}{2} \sum_{u,v \in V, u<v} |e^i_{uv} - e^j_{uv}|. \tag{3.3}$$

$e^i_{uv} = 1$ if edge e_{uv} is included in T_i and $e^i_{uv} = 0$ if not.

Like many other constrained graph problems, the OCST problem is $\mathcal{NP}$-hard (Garey and Johnson, 1979). Furthermore, since the problem is $\mathcal{MAX}$ $\mathcal{SNP}$-hard (Reshef, 1999), no polynomial-time approximation scheme exists, unless $\mathcal{NP} = \mathcal{P}$ (C. Papadimitriou and Yannakakis, 1988). Only for a few restricted problem instances algorithms exist, which return optimal solutions (Wu and K.-M. Chao, 2004). In addition, various approximation algorithms for the OCST problem have been developed (Peleg and Reshef, 1998; Wu, K.-M. Chao, and Tang, 2000a; Wu, K.-M. Chao, and Tang, 2000b). However, due to the $\mathcal{MAX}$ $\mathcal{SNP}$-hardness of the problem the solution quality of such approximation algorithms is very limited. To overcome the limitations of exact and approximation algorithms, many heuristics, especially EAs have been developed (Fischer and Merz, 2007; Y. Li and Bouchebaba, 2000; C. C. Palmer, 1994; Raidl and Julstrom, 2003a; Rothlauf, Goldberg, and Heinzl, 2002; Soak, 2006). For an overview of EAs for the OCST problem, we refer to Rothlauf (2006).

3.2.2 Experimental design

For finding optimal, or at least near-optimal solutions of OCST problems, we used a branch-and-bound algorithm (Ahuja and Murty, 1987a) for small problem instances with $n \leq 15$ and a GA for larger problem instances. The branch-and-bound algorithm is able to solve all problem instances with $n \leq 15$ in reasonable time.

The situation is different for larger problem instances ($n > 15$). Therefore, we used an iterative GA for such problems. Although GAs are heuristic search methods that cannot guarantee finding an optimal solution, we choose its design in such a way that we can assume that the found solution is optimal or near-optimal. We start

the iterative GA by applying a standard GA n_{iter} times to an OCST problem using a population size of N_0. T_0^{best} denotes the best solution that is found during the n_{iter} runs. In a next round, we apply again a GA n_{iter} times with $N_1 = 2N_0$ which finds the best solution T_1^{best}. We continue the iterations and double the population size $N_i = 2N_{i-1}$ until $T_i^{best} = T_{i-1}^{best}$ and $n(T_i^{best})/n_{iter} > 0.5$, this means T_i^{best} is found in more than 50% of the runs in round i. $n(T_i^{best})$ denotes the number of runs that find the best solution T_i^{best} in round i.

For the experiments, we use a standard, generational GA with crossover and mutation. Solutions are encoded using Network Random Keys (Rothlauf, Goldberg, and Heinzl, 2002), since GA performance is approximately independent of the structure of the optimal solution. The GA uses uniform crossover and tournament selection without replacement. The size of the tournament is three. The crossover probability is set to $p_{cross} = 0.7$ and the mutation probability (assigning a random value $[0, 1]$ to one allele) is set to $p_{mut} = 1/l$, where $l = n(n-1)2$.

To create random spanning trees, we create random Prüfer numbers (Prüfer, 1918) with uniform probabilities and transform the Prüfer numbers into spanning trees (Even, 1973, pp. 103-104), which is a bijective mapping. Therefore, random Prüfer numbers yield unbiased random spanning trees.

3.2.3 Properties of high-quality solutions

In this section, we analyze various properties of optimal solutions for Euclidean variants of OCST problems. If we observe a significant difference between optimal solutions and random solutions, we can use this knowledge to construct high-performance heuristic optimizations methods. Optimal solutions and random solutions are determined using the approaches discussed in Section 3.2.2. We always compare optimal solutions ('opt'), random solutions ('ran') and minimal spanning trees ('mst').

Distance weights and orientation

Rothlauf (2009a) analyzed properties of the OCST problem and showed that optimal solutions are biased towards the minimum spanning tree (MST). The distances between optimal solutions and MSTs are significantly smaller than the distances between optimal solutions and randomly generated solutions. Thus, using an optimization method that is biased towards MST-like solutions, the performance can be increased. Heuristic variants of edge-sets (Raidl, 2000; Raidl and Julstrom, 2003a) make use of this fact by favoring edges with a low distance weight (Rothlauf, 2009c).

In Steitz and Rothlauf (2008) we analyzed another property of edges in optimal solutions, the orientation of an edge. The orientation of an edge e_{ij} is the angle

$\gamma \in [0, 90]$ between e_{ij} and the line connecting the midway of e_{ij} and the center C of the tree. C is calculated as the average x-coordinates and y-coordinates of all nodes in the graph. Since $\gamma \leq 90$, the lower angle is chosen ($\gamma = min(\gamma_1, \gamma_2)$). For edges directly pointing to the center, $\gamma = 0$ holds. The experiments showed that edges pointing toward the graph's center are overrepresented in good solutions for Euclidean OCST instances. Using this knowledge in crossover operators allowed us to improve search performance of EAs.

Node degrees

In graph theory, the node degree $deg(v)$ is the number of edges connected to the vertex v in a graph G. $\delta(G)$ and $\sigma(G)$ denote the maximum and the minimum degree of a graph (Diestel, 2005, Sec. 1.2).

To analyze the node degree we created 1000 random OCST instances and determined optimal solutions using the approaches discussed in Sec. 3.2.2. The plots in Figure 3.1 display the results for Euclidean OCST instances.

The degree distributions of the three configurations vary considerably. For MSTs there are a lot nodes with degree two and maximal node degree of $\delta(G) = 4$. In the Euclidean case it is impossible for MSTs to have a node degree > 4 (Robins and Salowe, 1994). The highest node degrees are reached with optimal solutions. Here we have some nodes with very high node degrees. Comparing optimal and random solutions, we see that many more nodes (about 60 percent) in optimal solutions are leaf nodes ($deg(v) = 1$).

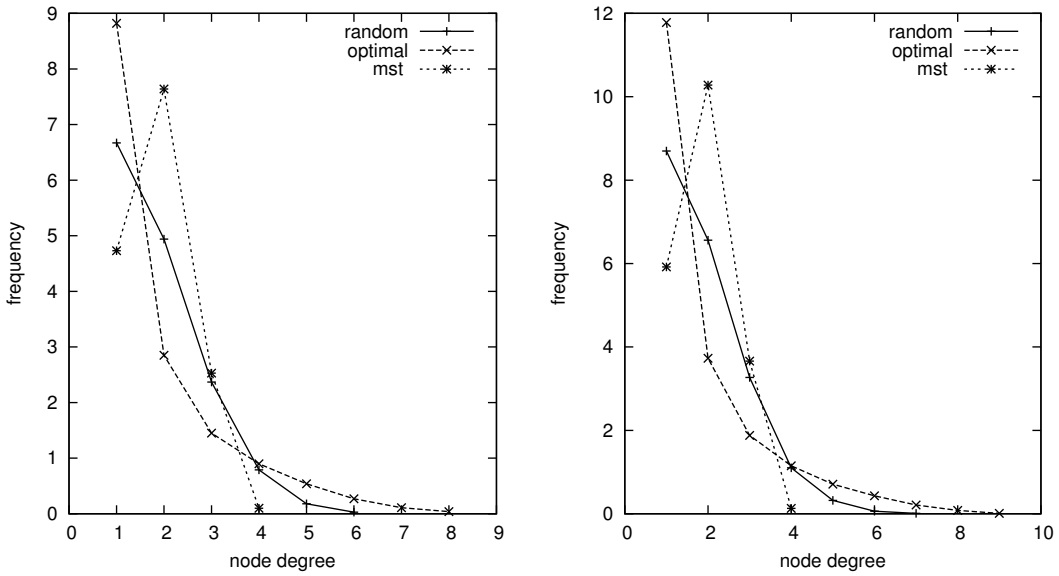
Location of nodes

In the last section, we have seen that in optimal solutions for Euclidean OCST instances, we have some nodes with very high node degrees. In contrast, we have many nodes with a degree of one. Now we analyze the locations of these two types of nodes for the same instances as in the previous section.

The plots in Figure 3.2 display the distance to the center over the node degree for instances with 15 and 20 nodes. The graph's center is calculated as the average X- and Y-coordinates. The distances are normalized using the maximal distance occurring in the tree. The node with distance 1 is the furthest node from the center.

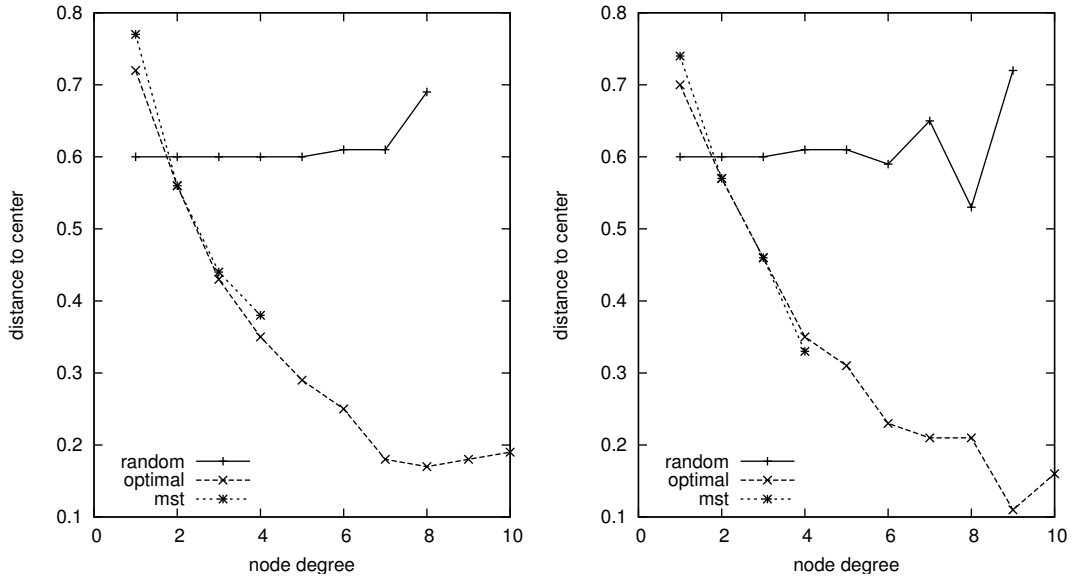
For optimal solutions and MSTs, nodes with higher degree are located closer to the center. The average distance decreases with increasing node degree. Since the maximal node degree for MSTs is four, the curve for MSTs terminates here. In contrast for random solutions, on average distances are the same for each node degree.

The results reveal that optimal solutions have a few nodes with very high degrees located near to the trees center. Additionally, the majority of the nodes have a



(a) $n = 15$ **(b)** $n = 20$

Figure 3.1: Node degree distribution of Euclidean OCST instances of different problem size n. The plots compare optimal solutions, random spaning trees and minimal spanning trees.



(a) $n = 15$ **(b)** $n = 20$

Figure 3.2: The plots show the average distances to the center over the node degree for optimal, random and minimum spanning trees. For optimal and MST, with larger distance to the center the node degree decreases. As we analyze Euclidean instances, MST have a maximum node degree of four.

degree of one and are located far away from the center. Therefore, to create high-quality solutions for the OCST problem, we have to create spanning trees with similar node degree distributions and make sure that nodes with high node degrees are located near to the graph's center.

Wiener index

The Wiener index is a topological index of a molecule used in chemical graph theory. It quantifies the degree of branching in a given tree. Paulden (2007) was the first to use the Wiener index in the context of EAs, to analyze properties of tree representations. A tree's Wiener index $WI(T)$ is defined as

$$WI(T) = \sum_{\epsilon \in E_T} n_1(\epsilon) \cdot n_2(\epsilon),$$

where $n_1(\epsilon)$ and $n_2(\epsilon)$ denote the number of nodes in the subtrees formed when edge ϵ is removed from T.

The larger the Wiener index of a tree, the smaller is the amount of branching. The extreme values for trees with n nodes are $WI = (n-1)^2$ for star trees and $WI = \binom{n+1}{3} = n(n+1)(n-1)/6$ for path trees. There exists a linear algorithm to compute the Wiener index (Aringhieri, Hansen, and Malucelli, 2001).

Table 3.2 shows the results of our empirical analysis of the Wiener index of Euclidean OCST instances with 10, 15 and 20 nodes. Comparing optimal and random solutions shows a significant difference between optimal and random solutions. On average the WI of optimal solutions is lower than the WI of random solutions. Therefore, optimal solutions are more star-like than random solutions.

In this section, we have analyzed different properties of Euclidean OCST instances. In summary, the following properties are known. Incorporating these while developing optimization methods, leads to a high performance.

- Edges with low weights and low orientation are preferred.
- Optimal solutions have some nodes with high node degrees. These hub-nodes are located next to the graph's center.
- Typically, more than 50 percent of the nodes are leaf nodes and are located far away from the center.
- Optimal solutions possess a low Wiener index, that is, they are star-like.

3.3 Constructing initial solutions for the OCST problem

In the last Section, we have shown that optimal solutions for Euclidean OCST instances have certain properties. We now present two construction heuristic which build spanning trees with similar probabilities. The first one is the Ahuja-Murty Tree Building Algorithm originally described in Ahuja and Murty (1987a), which relies on approximated costs and does not consider graph properties. The second heuristic approach only considers the graph properties.

3.3.1 Ahuja-Murty tree building

The tree construction heuristic (Ahuja and Murty, 1987a; Ahuja and Murty, 1987b) starts from a random seed node and iteratively appends edges to the partial solution. At the beginning all communication requirements are routed over the shortest paths in the completely connected graph G. Traffic between nodes of the partial solution are routed over the shortest paths in the partial solution. Adding new edges increases the communication costs, as more traffic is routed over the partial solution. New edges are chosen to be communication cost optimal. The heuristic requires to build all-pairs shortest paths in advance, which has a time complexity of $O(n^3)$. Since we are dealing with Euclidean instances only, finding all-pairs-shortest-paths is not neccessary, as the shortest path between any pair of adjacent nodes is the distance of the edge connecting the two nodes. Nevertheless the time complexity of Ahuja-Murty Tree Building is still $O(n^3)$. The original heuristic (denoted as AM-R) uses a random seeding node. We slightly modified the algorithm (denoted AM-C), which uses the node closest to the graph's center as seed.

3.3.2 A new problem-specific approach

Our new problem-specific approach builds a spanning tree from a seeding node s by repeatedly appending the lowest-weight edge that joins a new node to the growing tree. Choosing each new edge according to the modified weight w'_{ij} (equation 3.4), we prefer edges with low distance weight w_{ij} and low orientation γ_{ij}, as argued in Steitz and Rothlauf (2008). α and $\beta \in [0;1]$ are parameters that control the influence of w_{ij} and γ_{ij}. The distance weights as well as the orientation of edges are normalized using the maximum values $w_{max} = max(w_{ij})$ $(i,j = 1,\ldots,n)$ and $\gamma_{max} = max(\gamma_{ij})$ $(i,j = 1,\ldots,n)$. Therefore, $w'_{ij} \in [0,1]$. $dist_{(ij),C}$ denotes the distance of edge (i,j) to the graph's center C.

$$w'_{ij} = \begin{cases} \alpha \frac{w_{ij}}{w_{max}} + (1-\alpha)\frac{\gamma_{ij}}{\gamma_{max}} & \text{if } \frac{dist_{(i,j),C}}{dist_{max}} \geq \beta \\ w_{ij}/w_{max} & \text{otherwise} \end{cases} \tag{3.4}$$

The algorithm is designed to prefer edges with low distance weights and low orientation. Additionally, new nodes are always connected to the current partial tree. This behavior ensures that trees with certain node degree distributions are created. Algorithm 1 outlines the functionality of the approach. Paramters, besides the set of nodes V and edges E, are the weights w for each edge and the distance d of each node to the center. The function $ExtractMin(V, d)$ accesses and removes the element with minimal d from V. The algorithms starts by selecting the seeding node, which is the node closest to the center. Then iteratively the next node u is selected using $ExtractMin$. From the set of nodes connecting u and the partial solution already created, the edge (u, v) with minimal w' is selected and added to the partial solution. Once a complete spanning tree is created, the algorithm returns T.

Apparently, $ExtractMin$ has a time complexity of $O(n)$, therefore the overall time complexity of Algorithm 1 is $O(n^2)$.

Algorithm 1 ConstructGraph(V, E, w, d)

```
T ← ∅
ExtractMin(V, d)
while V ≠ ∅ do
    u = ExtractMin(V, d)
    choose edge {(u, v)} ∈ F with minimal w and v ∉ V
    T ∪ {(u, v)}
return T
```

3.4 Experimental results

To analyze the performance of the different construction heuristics, we perform various experiments. First we create solutions using initialization methods only and compare the quality of the found solutions. Then, we analyze the properties of the created spanning trees and investigate whether the trees have the same properties as high-quality solutions as shown in Section 3.2.3. Finally, we use inoculation to seed the initial population of a simple steady-state EA and analyze the performance.

3.4.1 Construction heuristics

In this first set of experiments we evaluate the quality of solutions for the OCST problem found by different construction heuristics. In particular, we compare the problem-specific approach (COH) presented in Sec. 3.3.2 and Ahuja-Murty Tree Building (Sec. 3.3.1).

We generate 1000 random Euclidean instances with different problem sizes $n = \{10, 15, 25, 50, 75, 100, 150, 200\}$ and compare the performance of the following six different initialization methods, as discussed in Section 3.3. For problem sizes $n = 10$ and 15 we determine optimal solutions for comparison using the approaches discussed earlier.

1. **random initialization (ran):** Non-heuristic initialization method.
2. **minimum spanning tree (mst):** Minimum spanning tree using Kruskal's algorithm.
3. **COH with random seed ($COH - R$):** COH heuristic using a random seed node.
4. **COH with center seed ($COH - C$):** COH heuristic using the center node as seed.
5. **Ahuja-Murty ($AM - R$):** Standard Ahuja-Murty tree building heuristic.
6. **Ahuja-Murty center ($AM - C$):** Ahuja-Murty initialization using center node as seed.

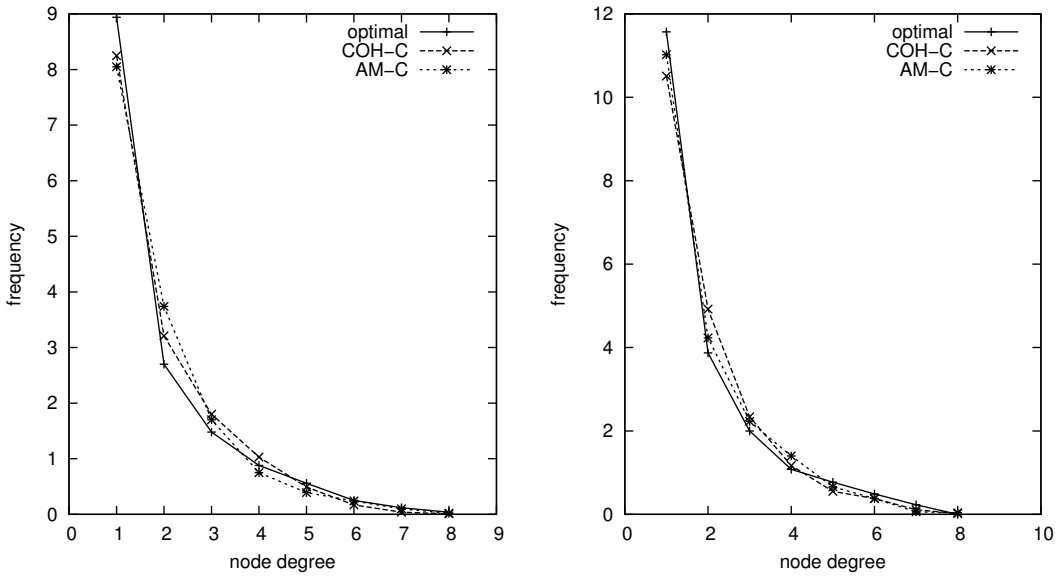
Table 3.1 shows the results of our experiments. $w(T_{best})$ denotes the average fitness of the created tree and σ the corresponding standard deviation. t_{cpu} is the average computation time (in seconds) for one tree construction. The table also lists the percentage of runs P_{suc} that find T_{opt} for instances with problem size $n \leq 15$.

The results show a high performance of the heuristic initialization methods, the average fitness values differ considerably compared to random tree construction. Confirming previous studies, MSTs are already good solutions. But the COH and AM variants perform significantly better. Especially the variants using the graph's center as seeding node show a high performance. The difference between these methods and MSTs gets larger with the problem size n. Comparing average CPU times, shows that the AM variants are more computational intensive.

Now we have a closer look at the generated solutions and analyze the properties of the created spanning trees. The plots in Figure 3.3 show the node degree distribution for $AM - C$ and $COH - C$. The charts look very similar for both methods. Comparing the distances to the center, Figure 3.4, shows that both construction

Table 3.1: Comparison of different construction heuristics

n		ran	mst	COH-R	COH-C	AM-R	AM-C
10	P_{suc}	0	0.01	0.05	0.12	0.27	0.25
	$w(T_{best})$	3363.48	1659.61	1635.51	1547.84	1519.23	**1519.7**
	σ	601.86	0	105.40	0	20.44	0
	t_{cpu}	0	0	0	0	0	0
15	P_{suc}	0	0	0.01	0.02	0.1	0.08
	$w(T_{best})$	9846.44	4152.43	4025	3708.18	3638.49	**3633.67**
	σ	0	0	0	0	0	0
	t_{cpu}	0	0	0	0	0	0
20	P_{suc}	0	0	0	0	0.09	0.09
	$w(T_{best})$	22013.25	8166.98	7897.78	6974.15	6860.46	**6793.52**
	σ	0	0	0	0	0	0
	t_{cpu}	0	0	0	0	0	0
25	$w(T_{best})$	37400	13084	12449	10944	10777	**10704**
	σ	5840.60	0	1336.71	0	200.23	0
	t_{cpu}	0	0	0	0	0	0
50	$w(T_{best})$	219151	60833	56089	45787	45288	**44635**
	σ	31984	0	8507	0	891	0
	t_{cpu}	0	0	0	0	0	0
75	$w(T_{best})$	604144	145893	133552	104799	102927	**101169**
	σ	85096	0	23287	0	1936	0
	t_{cpu}	0	0	0	0	0.01	0.01
100	$w(T_{best})$	1232013	274476	248813	188660	184417	**180798**
	σ	168682	0	47886	0	3471	00
	t_{cpu}	0	0	0	0	0.02	0.02
150	$w(T_{best})$	3356800	659053	599965	434379	420392	**410600**
	σ	451833	0	125409	0	7947	0
	t_{cpu}	0	0	0	0	0.09	0.09
200	$w(T_{best})$	6787512	1233670	1119350	782644	751030	**732414**
	σ	888959	0	257763	0	14077	0
	t_{cpu}	0	0.01	0.01	0.01	0.22	0.22

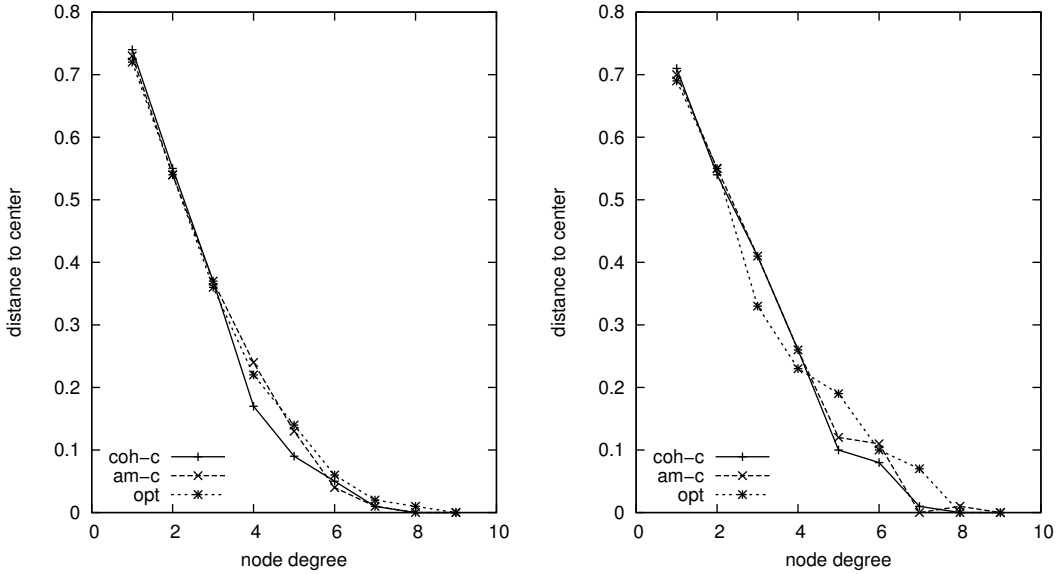


(a) $n = 15$ **(b)** $n = 20$

Figure 3.3: Node degree distribution of Euclidean OCST instances of different problem size n. The plots compare spanning trees created by initialization methods $COH - C$ and $AM - C$.

Table 3.2: Analysis of the Wiener index of spanning trees created by different initialization methods

n		ran	mst	opt	COH-R	COH-R	AM-R	AM-C
10	mean	128.63	142.1	112.44	123.96	114.02	118.32	118.96
	stdev	11.72	11.92	10.7	14.16	9.88	10.8	10.88
15	mean	379.22	443.82	309.98	361.73	320.4	328.97	330.26
	stdev	36.91	41.53	25.94	47.48	26.33	27.64	28.17
20	mean	805.74	992.91	643.81	795.19	658.75	672.34	667.11
	stdev	91.7	100.85	57.98	129.07	46.41	46.48	54.71



(a) $n = 15$ **(b)** $n = 20$

Figure 3.4: The plots show the average distances to the center over the node degree for spanning trees created using $AM - C$ and $COH - C$.

heuristics create spanning trees with similar distributions. For both the distance to the center decreases with higher node degrees. Therefore, the construction heuristics create spanning trees with similar node degree distributions as optimal solutions. Table 3.2 presents the analysis of the Wiener index of spanning trees created by initialization methods. Comparing the results to optimal solutions indicates that the Wiener index is also similar. Trees created by $COH - C$ exhibit a greater similarity to optimal solutions, the distance to the WI of optimal solutions is lower.

Overall, the results in this section indicate a high performance of the heuristic initialization methods for the OCST problem. On average the created spanning trees feature similar properties as optimal solutions.

3.4.2 GA

The final experiments compare the performance of simple steady-state EAs using different initialization methods. First, we study small problem instances where we determine optimal (or near-optimal) solutions as described in Section 3.2.2. Then, we examine larger problem instances with unknown optimal solutions.

Table 3.3: Performance of EAs with different initial solutions for small OCST instances

n		rnd	COH-R	COH-C	AM-R	AM-C
10	P_{suc}	0.72	0.78	0.79	**0.85**	**0.85**
	$w(T_{best})$	1456.48	1454.16	1454.04	1451.32	**1451.27**
	σ	5.52	4.29	2.81	3.36	1.92
	t_{cpu}	0.06	0.06	0.06	0.06	0.06
15	P_{suc}	0.25	0.36	0.41	**0.49**	**0.49**
	$w(T_{best})$	3539.98	3523.96	3519.51	**3508.66**	3508.84
	σ	28.95	21.78	16.27	15.82	10.64
	t_{cpu}	0.14	0.14	0.14	0.14	0.14
20	P_{suc}	0.04	0.09	0.15	**0.19**	0.16
	$w(T_{best})$	6399.33	6373.64	6356.13	6338.01	**6337.75**
	σ	50.83	42.40	27.60	35.83	21.10
	t_{cpu}	0.27	0.27	0.27	0.27	0.27

Small problem instances

To study the performance, we apply a simple steady-state EA using the initialization methods proposed in Section 3.3. For the experiments, we use uniform crossover and mutation. A population consists of $N = 50$ individuals. In each search step, one offspring is created by crossover (crossover probability $p_c = 1$) and mutation (mutation probability $p_m = 1/n$). The two parents are selected at random. If the cost of the offspring is lower than the cost of the worst individual in the population ($(w(T_{off}) \leq \max(w(T_i)$ for $i \in \{0, \ldots, N-1\})$), the offspring replaces the worst individual in the population. We present results for OCST instances of different sizes ($n = 10, 15$ and 20), where the optimal solutions are determined as described in Section 3.2.2. The EA terminates after $eval = 3,000$ fitness evaluations. For every problem size, 100 OCST instances are generated randomly and for each instance and configuration 20 EA runs are performed. We compare different initialization methods, as discussed in the previous experiments.

Table 3.3 lists the percentage P_{suc} of runs that find T_{opt}, the average costs $w(T_{best})$ of the best found solution, and the corresponding standard deviation σ. Additionally, the table shows the running time of the different configurations t_{cpu} in seconds.

The results indicate a high performance of EAs using heuristic initialization. Especially with the two Ahuja-Murty variants a high EA performance can be achieved. Comparing versions with random seed and the ones with center as seeding node, shows that using the graph's center as seed leads to a higher performance through-

Table 3.4: Number of fitness evaluations *eval* for large problem instances

n	50	75	100
eval	10,000	15,000	20,000

Table 3.5: Performance of EAs with different initial solutions for larger OCST instances

n		rnd	COH-R	COH-C	AM-R	AM-C
50	$w(T_{best})$	45551	44817	44523	44469	**44192**
	σ	710.71	416.70	213.93	398.55	142.63
	t_{cpu}	9.91	9.82	9.81	9.86	9.87
75	$w(T_{best})$	106154	103071	101774	101855	**100786**
	σ	2066.09	1135.78	507.58	1135.49	188.60
	t_{cpu}	46.65	45.99	45.92	46.34	46.35
100	$w(T_{best})$	196400	187866	185236	185485	**182539**
	σ	4973.52	2363.83	916.81	2168.84	239.59
	t_{cpu}	142.14	139.78	139.50	141.31	141.25

out all problem sizes. Comparing the average costs of the found solutions $w(T_{best})$ of AM and COH shows no significant difference. For these small instances the average CPU time is about identical for all variants.

Larger problem instances

In this section, we study the performance of the heuristic crossover operators for larger OCST instances with unknown optimal solution. The performance is measured by the costs of the best found solution.

We use the same EA as in the previous experiments. A population consists of $N = 50$ individuals and each EA run is stopped after *eval* fitness evaluations (see table 3.4). As EA performance usually increases with the number of fitness evaluations, we increase the number of fitness evaluations with larger n. We present results for OCST instances with different problem sizes ($n = 50, 75$ and 100). For each problem size, we create 100 random instances. The same initialization methods as in Section 3.4.2 are used and for each configuration and each problem instance we perform 10 EA runs.

Table 3.5 presents the average costs $w(T_{best})$ of the best found solution T_{best} over the 10 EA runs. Additionally, the standard deviation σ of the costs and the average running time t_{cpu} (in seconds) of one run is shown.

The results indicate a high performance of EAs using heuristic initialization, especially when using the graph's center as seeding node. EAs using $AM-C$ perform best for all problem sizes since the average total cost are always lowest. Furthermore, heuristic variants clearly outperform non-heuristic ones. Configurations $AM-C$ and $COH-C$ exhibit the lowest standard deviation σ, which indicates robust EA performance. Comparing these two configurations shows no big differences.

In summary, the experiments have shown that our simple construction heuristic creates spanning trees with similar properties as optimal solutions. Seeding the start-population with trees created by our initialization method improves EA performance for the OCST problem.

3.5 Summary and conclusions

This work studies the OCST problem and analyzes different graph properties of high-quality solutions. Results show that optimal solutions have a low Wiener index and are therefore star-like. More than half of the nodes are leaf-nodes and located far away of the center. In contrast, there are few nodes with high node degrees, these nodes are located next to the graph's center. Thus, EA performance can be systematically improved by biasing their search operators to construct similar graphs.

To show how these property of optimal solutions should be considered for the design of efficient EAs, we develop a simple construction heuristic. Analyzing the properties of the constructed spanning trees, shows that they possess similar properties as high-quality solutions. In a second step, we used the constructed solutions to seed a steady-state EA. The performance analysis for various test problems shows that search performance is increased compared to random start populations.

The results presented in this work suggest to use heuristic initialization which creates spanning trees with certain probabilities to improve EA performance for the OCST problem. Furthermore, this work shows that the most important aspect of designing high-performing EAs, is the incorporation of domain specific knowledge. While we focused on a certain graph problem in this work, the basic idea is generally applicable.

An interesting idea for future work is to develop a crossover or mutation operator similar to the proposed initialization methods. Another promising area to improve optimization methods for the OCST problem, is to analyze further graph properties of high-quality solutions and build problem-specific operators which consider the findings. Of course, the general approach, problem-specific EA design, can be applied to any graph problem or other hard optimization problem.

References

Ahuja, R. K. and V. V. S. Murty (1987a). "Exact and heuristic algorithms for the optimum communication spanning tree problem". In: *Transportation Science* 21.3, pp. 163–170.

Ahuja, R. K. and V. V. S. Murty (1987b). "New lower planes for the network design problem". In: *Networks* 17.2, pp. 113–127.

Aringhieri, R., P. Hansen, and F. Malucelli (2001). "A linear algorithm for the hyper-wiener index of chemical trees". In: *Journal of Chemical Information and Computer Sciences* 41.4, pp. 958–963.

Diestel, R. (2005). *Graph Theory*. 3rd ed. Graduate Texts in Mathematics 173. Heidelberg: Springer.

Even, S. (1973). *Algorithmic Combinatorics*. New York: The Macmillan Company.

Fischer, T. and P. Merz (2007). "A memetic algorithm for the optimum communication spanning tree problem". In: *Hybrid Metaheuristics*. Ed. by T. Bartz-Beielstein, M. J. B. Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels. Vol. 4771. Springer, pp. 170–184.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Hu, T. C. (1974). "Optimum communication spanning trees." In: *SIAM Journal on Computing* 3.3, pp. 188–195.

Li, Y. and Y. Bouchebaba (2000). "A new genetic algorithm for the optimal communication spanning tree problem". In: *AE '99: Selected Papers from the 4th European Conference on Artificial Evolution*. London, UK: Springer-Verlag, pp. 162–173.

Palmer, C. C. (1994). "An approach to a problem in network design using genetic algorithms". PhD thesis. Troy, NY, USA: Polytechnic University.

Papadimitriou, C. and M. Yannakakis (1988). "Optimization, approximation, and complexity classes". In: *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing, Chicago, Illinois, United States*. New York, NY, USA: ACM Press, pp. 229–234.

Paulden, T. J. (2007). "Combinatorial spanning tree representations for evolutionary algorithms". PhD thesis. Department of Computer Science, University of Exeter, UK.

Peleg, D. and E. Reshef (1998). "Deterministic polylog approximation for minimum communication spanning trees". In: *ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming*. London, UK: Springer-Verlag, pp. 670–681.

Prüfer, H. (1918). "Neuer Beweis eines Satzes über Permutationen". In: *Archiv für Mathematik und Physik* 27, pp. 742–744.

Raidl, G. R. (2000). "An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem". In: *Proc. of the 2000 Congress on Evolutionary Computation*. Piscataway, NJ: IEEE Service Center, pp. 104–111.

Raidl, G. R. and B. A. Julstrom (2003a). "Edge sets: an effective evolutionary coding of spanning trees". In: *IEEE Transactions on Evolutionary Computation* 7.3, pp. 225–239.

Reshef, E. (1999). "Approximating minimum communication cost spanning trees and related problems". MA thesis. Rehovot 76100, Israel: Feinberg Graduate School of the Weizmann Institute of Science.

Robins, G. and J. Salowe (1994). "On the maximum degree of minimum spanning trees". In: *Proceedings of the tenth annual symposium on Computational geometry*. ACM New York, NY, USA, pp. 250–258.

Rothlauf, F. (2006). *Representations for genetic and evolutionary algorithms (2. ed.)* Springer, pp. I–XVII, 1–325.

Rothlauf, F. (2009a). "An encoding in metaheuristics for the minimum communication spanning tree problem." In: *INFORMS Journal on Computing* 21.4, pp. 575–584.

Rothlauf, F. (2009c). "On the Bias and Performance of the Edge-Set Encoding". In: *IEEE Transactions on Evolutionary Computation* 13.3, pp. 486–499.

Rothlauf, F., D. E. Goldberg, and A. Heinzl (2002). "Network random keys: a tree representation scheme for genetic and evolutionary algorithms". In: *Evolutionary Computaton* 10.1, pp. 75–97.

Sharma, P. (2006). "Algorithms for the optimum communication spanning tree problem". In: *Annals of Operations Research* 143.1, pp. 203–209.

Soak, S.-M. (2006). "A new evolutionary approach for the optimal communication spanning tree problem". In: *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences* E89-A.10, pp. 2882–2893.

Steitz, W. and F. Rothlauf (2008). "Orientation matters: how to efficiently solve OCST problems with problem-specific EAs". In: *GECCO '08: Proceedings of*

the 10th annual conference on Genetic and evolutionary computation, Atlanta, GA, USA. New York, NY, USA: ACM, pp. 563–570.

Surry, P. D. and N. J. Radcliffe (1996). "Inoculation to initialise evolutionary search". In: *Selected Papers from AISB Workshop on Evolutionary Computing*. London, UK, UK: Springer-Verlag, pp. 269–285.

Wu, B. Y. and K.-M. Chao (2004). *Spanning Trees and Optimization Problems*. CRC Press.

Wu, B. Y., K.-M. Chao, and C. Y. Tang (2000a). "A polynomial time approximation scheme for optimal product-requirement communication spanning trees". In: *Journal of Algorithms* 36.2, pp. 182–204.

Wu, B. Y., K.-M. Chao, and C. Y. Tang (2000b). "Approximation algorithms for some optimum communication spanning tree problems". In: *Discrete Applied Mathematics* 102.3, pp. 245–266.

Chapter 4

Using penalties instead of rewards: solving OCST problems with guided local search

Wolfgang Steitz, Franz Rothlauf

Abstract

This paper considers the optimal communication spanning tree (OCST) problem. Previous work analyzed features of high-quality solutions and found that edges in optimal solutions have low weight and point towards the center of a tree. Consequently, integrating this problem-specific knowledge into a metaheuristic increases its performance for the OCST problem. In this paper, we present a guided local search (GLS) approach which dynamically changes the objective function to guide the search process into promising areas. In contrast to traditional approaches which reward promising solution features by favoring edges with low weights pointing towards the tree's center, our GLS penalizes low-quality edges with large weights that do not point towards the tree's center. Experiments show that GLS is a powerful optimization method for OCST problems and outperforms standard EA approaches with state-of-the-art search operators for larger problem instances. However, GLS performance does not increase if more knowledge about low-quality solutions is considered but is about independent of whether edges with high weight or wrong orientation are penalized. This is in contrast to the classical assumption that considering more problem-specific knowledge about high-quality solutions does increase search performance.

4.1 Introduction

The optimal communication spanning tree (OCST) problem (Hu, 1974) is a common $\mathcal{NP}$-hard combinatorial optimization problem which seeks a spanning tree that satisfies given communication requirements and leads to minimal total costs. Researchers studied various exact, approximate and heuristic solution approaches for the problem. The current state-of-the-art approaches (Fischer and Merz, 2007; Raidl and Julstrom, 2003a; Rothlauf, 2009b) are based on heuristics and metaheuristics, in particular evolutionary algorithms (EA).

Previous work (Rothlauf, 2009b; Steitz and Rothlauf, 2008, Chapter 3) studied the properties of OCST problems. They found that edges with low distance weights pointing towards the center of a graph occur with higher probability in high-quality solutions. Additionally, optimal solutions are star-like, nodes with a large node degree are near the tree center, and nodes with small node degrees are far away from the center. Furthermore, in optimal solutions, more than half of the nodes are leaf nodes with a node degree of only one. There are several ways in which such a-priori knowledge can be exploited in the design of efficient metaheuristics for OCST problems (Rothlauf, 2009b):

1. Initialize metaheuristic with solutions that have similar properties as high-quality solutions: Chapter 3 presented a simple initialization method which creates solutions with the desired tree properties. The solutions are used in the initial population of a genetic algorithm, which improved the overall search performance.

2. Use a redundant representation that over-represents solutions with certain properties: Rothlauf (2009a) applied the idea of using redundant representations that over-represent high-quality building blocks to the link-biased (LB) encoding. The LB encoding encodes trees similar to MSTs with higher probability. Experiments show that using the LB encoding improves search performance.

3. Use search operators that favor building blocks, similar to those of known good solutions: examples for this approach are the heuristic crossover and mutation operators of the edge-set (ES) encoding (Raidl and Julstrom, 2003a; Rothlauf, 2009c). Both crossover and mutation favor low-weight edges when creating offspring solutions, which leads to a performance increase in comparison to non-heuristic operators. Similarly, the search operators of ES can be extended to include edge weights *and* edge orientation (Chapter 2).

4. Assign higher fitness values to solutions with similar characteristics as high-quality solutions: we are unaware of any approach for the OCST problem that

modifies the fitness function according to the characteristics of high-quality solutions.

An approach to dynamically change the objective function in order to lead heuristic search to promising areas is guided local search (GLS) (Voudouris, 2003; Voudouris and Tsang, 1999). To apply GLS, it is necessary to define solution features and their corresponding costs. Natural features are building blocks of a solution. For a graph problem, these are typically the edges. Corresponding costs can depend on the weight and orientation of edges. During the search process, some low-quality solution features are penalized which helps local search to escape local optima. In this paper, we apply GLS to the OCST problem and show how problem-specific knowledge can be used to improve the search performance by changing the evaluation function. The main findings of this paper are:

1. Problem-specific GLS outperforms EAs using state-of-the-art search operators which also consider the weights and orientation of edges.

2. The performance of GLS is about independent of how much problem-specific knowledge about the problem is considered. Thus, GLS shows similar behavior if edges with high weight, wrong orientation, or both, are penalized. This is in contrast to results for EAs where the choice of problem-specific knowledge has a strong influence on search performance.

3. GLS as well as EAs show similar behavior for different types of OCST problems (clustered versus non-clustered problem instances)

The next section defines the OCST problem, gives an overview of existing solution strategies and summarizes the knowledge about high-quality solutions. Section 4.3 presents GLS and develops a problem-specific GLS approach for the OCST problem. Section 4.4 presents experimental results for different types of OCST instances. The paper ends with concluding remarks.

4.2 The OCST problem

4.2.1 Definition

The optimal communication spanning tree (OCST) problem is a common combinatorial tree optimization problem, which was first described by Hu (1974). Given a collection of nodes and the distances and communication demands between them, we seek a tree that connects all nodes at minimum total communication cost, defined to be the sum of the products of the distances and communication demands between all pairs of nodes.

Formally, let $G = (V, E)$ be a complete, weighted, undirected graph with $n = |V|$ nodes and $m = |E|$ edges. Communication or transport requirements are given a priori in an $n \times n$ *demand matrix* $R = (r_{ij})$. Analogously, an $n \times n$ *distance weight matrix* $W = (w_{ij})$ specifies distance weights. The solution quality or cost $f(T)$ of a tree $T = (V, F)$ with $F \subseteq E$ and $|F| = n - 1$ is calculated as:

$$f(T) = \sum_{i,j \in V} r_{ij} \cdot pl_{ij}, \tag{4.1}$$

where pl_{ij} denotes the path length between nodes i and j which is calculated as the sum of the weights of all edges on the path between i and j in T. It depends on the structure of T and the distance matrix W. For spanning trees, there exists only one unique path between any pair of nodes and the path length pl can be determined using a depth first search. T is the optimal communication spanning tree if $f(T) \leq f(T')$ for all other spanning trees $f(T')$.

As many other constrained graph problems, the OCST problem is $\mathcal{NP}$-hard (Garey and Johnson, 1979). Furthermore, the problem is $\mathcal{MAX}$ $\mathcal{SNP}$-hard and no polynomial-time approximation scheme exists, unless $\mathcal{NP} = \mathcal{P}$ (C. H. Papadimitriou and Yannakakis, 1991). Hu (1974) presented two simplifications of the problem which are polynomially solvable. For the general case of the OCST problem, Ahuja and Murty (1987a) proposed an exact approach based on branch-and-bound and Contreras, Fernández, and Marín (2010) formulated an integer programming problem. However, these exact approaches only work for small problem instances.

In addition, various approximation algorithms for the OCST problem have been developed (Peleg and Reshef, 1998; Sharma, 2006; Wu, K.-M. Chao, and Tang, 2000a), although, due to the $\mathcal{MAX}$ $\mathcal{SNP}$-hardness of the problem, the solution quality of such approximation algorithms is very limited. To overcome the limitations of exact and approximation algorithms, many heuristics, in particular EAs have been developed (Carrano et al., 2010; Fischer and Merz, 2007; C. Palmer and Kershenbaum, 1995; Raidl and Julstrom, 2003a; Rothlauf, 2009b; Soak, 2006).

4.2.2 Properties of high-quality solutions

Rothlauf (2009b) studied the properties of OCST problems and found that optimal solutions are biased towards minimum spanning trees (MST). The distances between optimal solutions and MSTs are significantly lower than the distances between optimal solutions and randomly generated solutions. Thus, by using optimization methods that are biased towards MST-like solutions, search performance can be increased. Heuristic variants of edge-sets (Raidl and Julstrom, 2003a) make use of this fact by favoring edges with small distance weights.

Chapter 2 analyzed the orientation of edges in high-quality solutions for Euclidean OCST instances. The orientation of an edge is low if the edge points towards the

tree's center. Edges with small orientation are over-represented in optimal solutions. The larger the edge orientation, the lower the probability that this edge is part of an optimal solution. Therefore, biasing search operators to favor edges with low weight *and* low edge orientation leads to a performance increase.

4.3 Guided local search

To escape local optima, local as well as evolutionary search algorithms use a wide range of diversification mechanisms, like tabu lists (used in Tabu Search) or the acceptance of worse solutions with some probability (used in simulated annealing). Guided local search (GLS), which was presented by Voudouris in 1997 (Voudouris, 1997; Voudouris, 2003; Voudouris and Tsang, 1999), ensures diversification by dynamically changing the objective function: it adds penalties based on the knowledge gained during the search. By dynamically modifying the fitness function, the search does not get stuck in local optima, but rather gains the ability to leave local optimum and to explore other areas in the search space. Usually, the penalties take into account the information of previous search steps, but they can also incorporate prior knowledge of the problem. GLS is easy to implement and apply, as there are only a few parameters.

GLS has been successfully applied to a wide range of combinatorial optimization problems, such as the traveling salesman problem (Voudouris and Tsang, 1999), function optimization, partial constraint satisfaction, vehicle routing, and others (Voudouris, 2003). This section outlines the functionality of the algorithm and demonstrates the design of a problem-specific GLS variant for OCST problems.

4.3.1 Guided local search algorithm

GLS's mechanism of changing the objective function is based on *solution features*, which represent solution properties. A solution feature is a non-trivial property of a solution and allows to distinguish between different solutions. If GLS reaches a local optimum, the worst solution feature is penalized. Penalizing solution features alters the objective function and directs the search to other areas of the search space.

As GLS penalizes solution features, it modifies the original evaluation function f. Consequently, the objective function h used by GLS is defined as

$$h(T) = f(T) + \lambda \cdot \sum_{i=1}^{M} p_i \cdot I_i(T), \tag{4.2}$$

where T is the tree to be evaluated, f is the original evaluation function with no modifications (equation 4.1), M is the number of solution features, p_i is the penalty

for solution feature i, and parameter λ controls the importance of the penalties. The indicator function $I_i(T)$ tells us whether or not feature i is part of a solution T:

$$I_i(T) = \begin{cases} 1, & \text{if feature } i \text{ is in solution } T, \\ 0, & \text{otherwise} \end{cases}.$$

Tsang, Mills, and Ford (2002) suggested to set the parameter λ according to the first found local optimum T_1 as

$$\lambda = \alpha \frac{f(T_1)}{M},$$

with $\alpha \geq 0$. This allows us to set the parameter α independently of the problem instance (Tsang, Mills, and Ford, 2002; Voudouris, 2003).

Algorithm 2 GuidedLocalSearch(V, E, w)

```
k ← 0
T_0 ← generateRandomTree()
for i = 1 to M do
    p_i ← 0
while stopping condition is not met do
    T_{k+1} ← LocalSearch(T_k)
    for i = 1 to M do
        util_i(T_{k+1}) ← I_i(T_{k+1}) · c_i/(1 + p_i)
    for all i where util_i is maximum do
        p_i ← p_i + 1
    k ← k + 1
return overall best solution
```

Algorithm 1 outlines the functionality of GLS. At the beginning, a random initial solution T_0 is created and all penalties p_i are set to zero. After the initialization phase, a local search is iteratively performed. Local search uses the evaluation function h (see (4.2)), starts with T_k, and returns the local optimum T_{k+1}. Then, the worst solution feature of the returned local optimum is penalized. To identify the worst solution feature, the utility of penalization $util_i(T_{k+1})$ for each solution feature p_i is calculated as

$$util_i(T_{k+1}) = I_i(T_{k+1}) \cdot \frac{c_i}{1 + p_i},$$

where c_i quantifies the cost if feature f_i is part of a solution T. Usually, the c_i are constant during a GLS run. Then, the solution feature i with the highest utility of

penalization $\max_i(util_i(T_{k+1}))$ is penalized by increasing p_i by one ($p_i \leftarrow p_i + 1$). This feature with the highest utility of penalization, causes the highest cost. If there are more than one solution feature with the same maximal value, all of them are penalized. The higher the cost c_i of a solution feature is, the larger the utility of penalizing this feature is. To ensure diversification and to avoid that always the same solution feature is penalized, the current penalties are also part of the utility function.

Then, the next iteration starts from the current local optimum. Since the evaluation function is altered, the current solution is not necessarily a local optimum any-more and local search is able to escape from the local optimum and explore new solutions. If the stopping condition, e.g., a given number of iterations, is reached, the algorithm terminates and returns the overall best solution.

4.3.2 Solving the OCST problem using GLS

As mentioned by Voudouris (2003), edges are proper solution features for graph problems. Therefore, we define the set of solution features equal to all edges E in the fully connected graph. Then, the number of solution features M that are part of a solution T is $n-1$. For random instances of OCST problems (see Section 4.4.1), where the distance weights w_{ij} are chosen randomly, proper feature costs c_{ij} are the distance weights w_{ij}.

For Euclidean OCST problem instances, where the nodes are set on a two-di/-men/-sional plane and the distance weights w_{ij} are the Euclidean distances between the nodes, the angle γ_{ij} denotes the orientation of edge e_{ij} ($\gamma_{ij} = 0$ if edge e_{ij} points towards the center of the graph and $\gamma_{ij} = 90$ if edge e_{ij} is perpendicular to the line connecting the middle of e_{ij} and the graph's center). Following Steitz and Rothlauf (2008), the quality of an edge depends not only on its distance weight w_{ij}, but also on its orientation γ_{ij}. Therefore, we distinguish between three possible types of cost:

1. If we use the Euclidean distances between nodes i and j, we obtain $c^w_{ij} = w_{ij}$.
2. If we consider only the orientation, we obtain

$$c^o_{ij} = \begin{cases} \gamma^*_{ij} & \text{if } dist^*_{(i,j),C} \geq 0.3 \\ w^*_{ij} & \text{otherwise.} \end{cases} \tag{4.3}$$

3. If we consider distance weight *and* orientation, Steitz and Rothlauf (2008) recommend using

$$c^{ow}_{ij} = \begin{cases} 0.7w^*_{ij} + 0.3\gamma^*_{ij} & \text{if } dist^*_{(i,j),C} \geq 0.3 \\ w^*_{ij} & \text{otherwise.} \end{cases} \tag{4.4}$$

$dist_{(ij),C}$ denotes the Euclidean distance between the middle of edge $e_{i,j}$ and the graph's center C. "*" indicates that w_{ij}, γ_{ij}, and $dist_{(ij),C}$ are normalized with the maximum values $w_{\max} = \max_{e_{ij} \in E}(w_{ij})$, $\gamma_{\max} = \max_{e_{ij} \in E}(\gamma_{ij})$, and $dist_{e_{ij},C,\max} = \max_{e_{ij} \in E}(dist_{(ij),C})$. Therefore, $c_{ij} \in [0, 1]$.

Following Steitz and Rothlauf (2008), edge orientation is only considered for edges that are some distance away ($dist^*_{(i,j),C} \geq 0.3$) from the graph's center. This is necessary, as edge orientation is meaningless for edges next to the graph center. In contrast to c^w_{ij} and c^o_{ij}, which only consider either distance weights or edge orientation, c^{ow}_{ij} considers both weight *and* orientation.

The local search LocalSearch(T_k) starts with T_k and iteratively examines a randomly selected neighboring tree. If a neighboring tree has a lower objective value h than the original solution, it replaces the original tree. The neighborhood of a tree consists of all trees that can be created by changing two edges. To generate neighboring trees, the insert-before-deletion operator from Raidl and Julstrom (2003a) is used. This local search operator chooses a random edge not yet included in the tree and inserts it, creating a cycle. To restore a feasible tree, the operator identifies the edges forming the cycle and removes one of those edges at random. Cycle detection is done using a depth-first search and the running time is $O(n)$. Raidl and Julstrom (2003a) also proposed a heuristic variant of the local search operator which prefers edges with lower weight when inserting new edges. The number of different neighbors of a tree depends on the tree's structure and varies between $(n-1)(n-2)$ and $\frac{1}{6}n(n-1)(n+1) - n + 1$. If there is no improvement for l consecutive search steps, the search process is likely stuck in a local optimum. At this point, the local search is stopped and the solution feature with highest $util_i$ is penalized.

To ensure that the final solution returned by GLS is a local optimum according to the original fitness function f, the original objective function $f(T)$ is used for the very last 500 evaluations of an GLS run.

The only GLS parameter requiring tuning is $\alpha \in]0, 1]$. For the OCST problem, we tried different settings and obtained the best results for $\alpha = 0.3$. This parameter setting was also recommended by Tsang, Mills, and Ford (2002), who studied other combinatorial optimization problems.

4.4 Experimental results

This section describes random, Euclidean and clustered OCST problem instances and studies the performance of local search, GLS, and EAs using edge-sets.

4.4.1 Problem instances

We follow previous work (Raidl and Julstrom, 2003a; Rothlauf, 2009b) and use randomly created OCST test instances. The real-valued demands r_{ij} are randomly created and uniformly distributed in $]0, 10]$. For the random instances, the real-valued distance weights w_{ij} are chosen at random in the interval $[0, 10]$. For the Euclidean instances, the distance weights w_{ij} are calculated as the Euclidean distances between the nodes i and j which are randomly placed on a 10×10 2-dimensional grid.

Clustered instances are created by choosing c cluster centers by randomly placing them on a 10×10 2-dimensional grid. The n nodes are split into c cluster. For each node, a Gaussian random variable with a deviation of $1/(c \cdot tightness)$ specifies the distance between the node and the center of the cluster. The parameter *tightness* controls how far away the nodes are from the cluster's center. In our experiments we set $tightness = 0.5$ and create $\lfloor \sqrt{n} + 0.5 \rfloor$ clusters.

For each problem type, we create instances of different sizes (n=10, 15, 25, 50, 75, 100). For every problem size $n < 75$, we create 1000 random instances. Due to long running times, we only create 100 instances for $n \geq 75$. Overall, we create 12,600 different OCST instances.

4.4.2 Experimental setup

We compare the GLS approach to a simple local search and to the state-of-the art EA using edge-sets and different recombination operators. The simple local search is used as a baseline.

Local search (denoted LS) is equivalent to LocalSearch(T_0) (see Section 4.3.2), where T_0 is a random initial solution. In each search step, it examines a random neighboring tree. If the new solution has lower cost than the original tree, it replaces the original tree. Like the other search methods, LS is terminated after *eval* fitness evaluations (see Table 4.1).

The steady-state EA uses edge-sets (ES) and the search operators proposed by Raidl and Julstrom (2003a) and Fischer and Merz (2007). The population size is set to $N = 200$. In each search step, a new solution T_{off} is created by recombination and mutation. Mutation is applied after crossover with a probability $p_m = 1/n$ (Raidl and Julstrom, 2003a). The resulting offspring T_{off} is inserted into the population *pop* if it has lower cost than the worst solution in the population ($f(T_{off}) < min_{T \in pop} f(T)$). The initial population consists of random spanning trees. The EA terminates after *eval* fitness evaluations.

For mutation, the insert-before-deletion operator (Raidl and Julstrom, 2003a) is used. We use different types of crossover operators. The operators proposed by Raidl and Julstrom (2003a) create an offspring from two parental trees by first adding all common edges of both parents into the offspring. In a second step, the offspring is

Table 4.1: Maximum number of *eval* evaluations for different problem sizes

n	10	15	25	50	75	100
eval	2,000	3,000	5,000	10,000	20,000	40,000

completed using only edges from the parents. Variants of this crossover differ in the edge selection strategy, which determines the edges to be inserted into the offspring tree. We also implemented the path merging (PM) recombination operator proposed by Fischer and Merz (2007). Here, entire paths are selected from the parents and included in the offspring. The offspring also consists of parental edges only.

We extend the original crossover operators, which only consider edge weights, and consider edge weights *and* edge orientations as proposed in Chapter 2 for Euclidean and clustered instances. Due to the bias of the crossover operators, edges with certain properties are preferred.

The functionality of GLS is described in Section 4.3.2. Initial solutions are created at random. For random problem instances, the cost of a solution feature (edge) are the edge weights ($c_{ij} = w_{ij}$). For Euclidean and clustered instances, we either use c_{ij}^{w} (edge weights), c_{ij}^{o} (edge orientation), or c_{ij}^{ow} (weight and orientation) as the cost c_{ij} of a solution feature. The last 500 search steps of each GLS run are performed using the original cost function f to obtain a local optimum solution.

Each optimization run is terminated after *eval* fitness evaluations. Since a larger number of evaluations increases search performance, we also increase *eval* for larger n (see Table 4.1). For each OCST problem instance, ten optimization runs are performed. For small instances ($n = 10$), a branch-and-bound algorithm (Ahuja and Murty, 1987a) is able to determine an optimal solution in a reasonable amount of time. P_{suc} indicates the percentage of runs of a metaheuristic that finds the optimal solution.

4.4.3 Random instances

Table 4.2 presents results for the random test distances. We show the percentage P_{suc} of runs that find the optimal solution (only for $n = 10$), the mean cost $\mu(f(T))$ of the best solution at the end of a run and the corresponding standard deviation σ. We compare an EA using edge-sets (ES), local search (LS), and three different variants of GLS with $l = 100$, $l = 200$ and $l = 300$. l specifies the maximal number of non-improving search steps, until the current solution is declared a local optimum and LocalSearch() is stopped.

We observe that ES and GLS clearly outperform the simple local search. The difference is smaller for small instances, since these instances are relatively easy

Table 4.2: Performance comparison for OCST instances with random distance weights

n		ES	GLS 50	GLS 100	GLS 300	LS
10	P_{suc}	0.9	0.67	0.76	**0.94**	0.9
	$\mu(f(T))$	703.44	712.8	708.85	**703.36**	706.79
	σ	0.0	0.0	0.0	0.0	0.0
15	$\mu(f(T))$	**1336.53**	1368.03	1371.48	1342.3	1360.38
	σ	0.0	54.96	53.35	0.0	0.0
25	$\mu(f(T))$	2802.77	**2791.16**	2876.09	2878.88	3026.24
	σ	0.0	110.84	155.21	141.41	284.35
50	$\mu(f(T))$	8086.99	**7573.48**	7643.91	8124.03	9754.27
	σ	521.18	530.9	548.72	643.6	1475.73
75	$\mu(f(T))$	15779.15	**13665.58**	13744.49	14409.58	21290.05
	σ	1223.09	1060.07	1133.78	1324.06	3784.91
100	$\mu(f(T))$	23538.09	19152.1	**19127.22**	19470.74	36611.01
	σ	1806.64	1528.39	1490.94	1597.45	6974.1

to solve. Comparing GLS and ES indicates a high performance of the GLS approach. For all problem sizes, all GLS configurations are able to find solutions with significantly lower costs. The setting of l has no significant influence on the GLS performance, but setting $n = 100$ seems to be a good compromise.

Statistical tests confirm these observations. For $n \geq 50$ the differences between ES, LS and all GLS configurations are significant using a pairwise ranked t-test with an error level of $p < 0.0001$. Also, differences between the three GLS configurations are not statistically significant.

4.4.4 Euclidean instances

For Euclidean test instances, we study three different variants of GLS using different costs of the solutions features: the first variant (GLS-w) uses only edge weights c_{ij}^w as costs of solution features, the second one (GLS-o) uses only edge orientation c_{ij}^o, and the third one (GLS-ow) uses edge weight and edge orientation c_{ij}^{ow}. For all GLS configurations, $l = 100$.

We analogously design the EA using edge-sets. We have heuristic variants, where the heuristic crossover operator uses either only edge weights c_{ij}^w (hES-w), only orientation c_{ij}^o (hES-o), or edge weights and edge orientation c_{ij}^{ow} (hES-ow). In addition, we have a naive non-heuristic variant (Raidl and Julstrom, 2003a), which does not use any problem-specific knowledge and is not biased towards low-weight

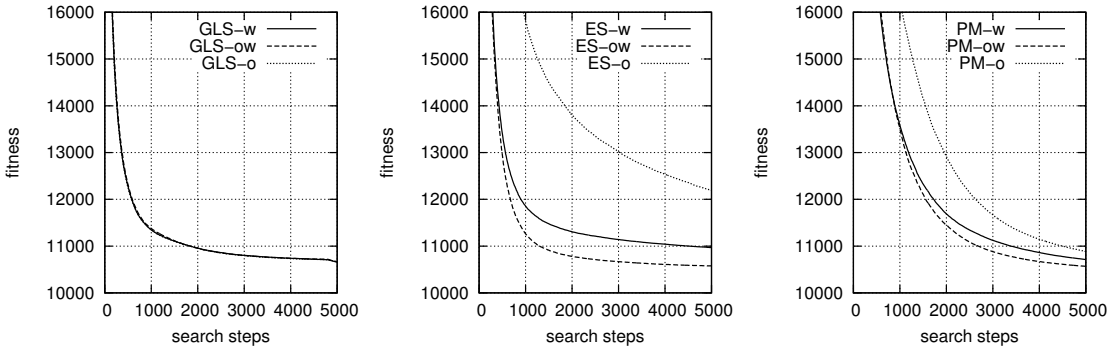
Table 4.3: Performance comparison for Euclidean OCST instances

n		ES	hES-w	hES-o	hES-ow	PM-w	PM-o	PM-ow	GLS-w	GLS-o	GLS-ow
10	P_{suc}	0.52	0.56	0.19	0.74	0.76	0.59	**0.81**	0.6	0.57	0.57
	$\mu(f(T))$	1497.56	1496.13	1537.7	1490.86	1490.56	1494.34	**1489.22**	1498.84	1500.45	1500.29
	σ	12.82	6.47	21.72	3.51	5.2	9.03	3.98	17.59	17.95	17.63
15	$\mu(f(T))$	3717.48	3647.52	3861.78	3594.73	3606.18	3633.48	**3588.57**	3629.44	3634.22	3634.42
	σ	86.84	31.32	79.64	15.6	31.79	45.74	23.78	55.53	56.02	55.08
25	$\mu(f(T))$	11911	11032	12272	10639	10770	10924	**10628**	10727	10716	10717
	σ	542.54	150.79	410.92	77.31	157.01	200.3	114.33	197.22	170.61	169.87
50	$\mu(f(T))$	60623	48373	59281	45484	46542	48051	45477	45244	**45206**	45227
	σ	5693.67	998.49	3114.63	652.19	889.9	1387.67	670.22	1033.07	971.01	992.63
75	$\mu(f(T))$	131587	111396	137231	104231	106140	109295	103847	102958	102838	**102804**
	σ	10402	2374.64	6951.0	1457.22	1805.59	2916.69	1353.33	2384.51	2016.76	2145.68
100	$\mu(f(T))$	214722	199022	233136	187165	188690	192228	185705	184815	**184676**	184825
	σ	11815	3792.53	9366.03	2507.34	3033.63	3641.53	2146.13	4095.18	3834.54	3663.27

edges (denoted as ES). Another three EA configurations use path merging recombination (PM) proposed by Fischer and Merz (2007). This crossover identifies low weight paths in the parents and includes them in the offspring. In our version, the path lengths are either determined by edge weights c_{ij}^{w} (PM-w), edge orientation c_{ij}^{o} (PM-o), or edge weight and orientation c_{ij}^{ow} (PM-ow).

Some of the configurations represent the state-of-the-art for OCST problems with Euclidean distances and were introduced in previous work. hES-w is the best performing configuration in Rothlauf (2009c). Chapter 2 proposed hES-ow and showed its superiority in comparison to hES-w. Fischer and Merz (2007) proposed the PM-w crossover as the best performing crossover operator for Euclidean OCST instances.

Table 4.3 presents the results. We show P_{suc} (only for $n = 10$), the mean cost $\mu(f(T))$ of the best solution at the end of a run and the corresponding standard deviation σ. Again, we observe that all GLS variants outperform the EA approaches for larger problem instances ($n \geq 50$). For smaller instances, the EAs are slightly better. Comparing the different EA configurations, we find that considering problem-specific knowledge for the crossover operator improves search performance. The more knowledge is integrated, the better the results are. ES-ow is significantly better than ES-o or ES-w. The same applies to PM-ow, which significantly outperforms PM-o and PM-w. Furthermore, results are significantly better if we consider only edge weights (-w) in comparison to if we consider only edge orientation (-o). This confirms previous results, that edge weights are more important than edge orientation (Chapters 2 and 3). Combining both properties of high quality edges – namely low weights and



(a) GLS **(b)** ES **(c)** PM

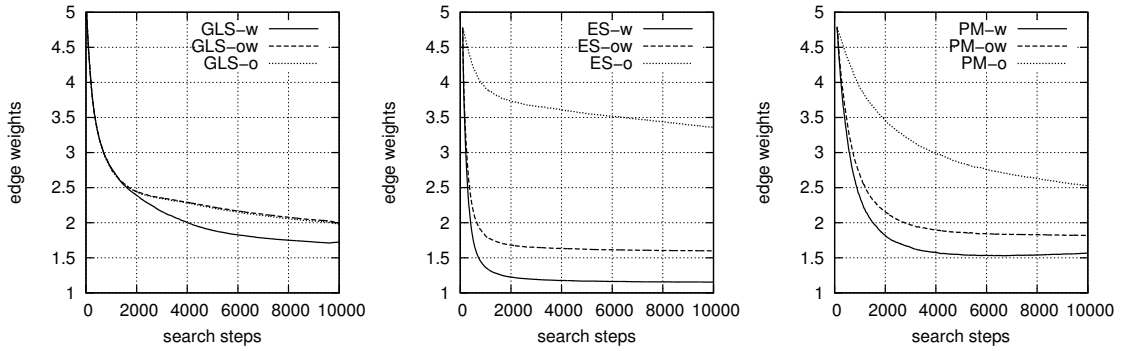
Figure 4.1: We show the average fitness of the current best solution over the number of search steps for different variants and Euclidean instances ($n = 25$). All GLS variants show very similar behavior. In contrast, the ES and PM variants differ significantly and the configurations considering both weight and orientation perform the best.

the characteristic of pointing towards the center of a graph (-ow) – leads to best results. A pairwise ranked t-test confirms these results. For $n \geq 50$, comparing the ES configurations all p-values are $p < 0.0001$. For PM, the differences are significant for $n >\geq 50$ with p-values $p < 0.02$, except for PM-w vs PM-ow and $n = 75$ ($p = 0.699$).

The situation is different for GLS. The average fitness $\mu(f(T))$ at the end of a run shows no significant difference whether we use edge weights, edge orientation or both as costs of solution features; $p > 0.35$ using a pairwise ranked t-test for all problem sizes.

The main difference between the EAs and GLS is the type of additional selection pressure that is introduced by considering problem-specific knowledge. ES and PM favor edges with low weight and proper orientation by giving them a higher chance to be selected in recombination. In contrast, GLS penalizes low-quality solution features and does not reward "good" solution features. Since all three types of feature costs (weight, orientation, and both combined) are able to identify low-quality solutions features, we observe no performance difference. We study this conjecture in the further paragraphs.

Figure 4.1 plots the average fitness of the current best solution over the number of search steps for $n = 25$. Results for other tree sizes are similar. The fitness values are averaged over all runs and all instances. For ES, there are no results for the first $M = 200$ evaluations, since these are necessary for evaluating the initial population. As expected, the GLS variants progress faster to high-quality solutions, since they



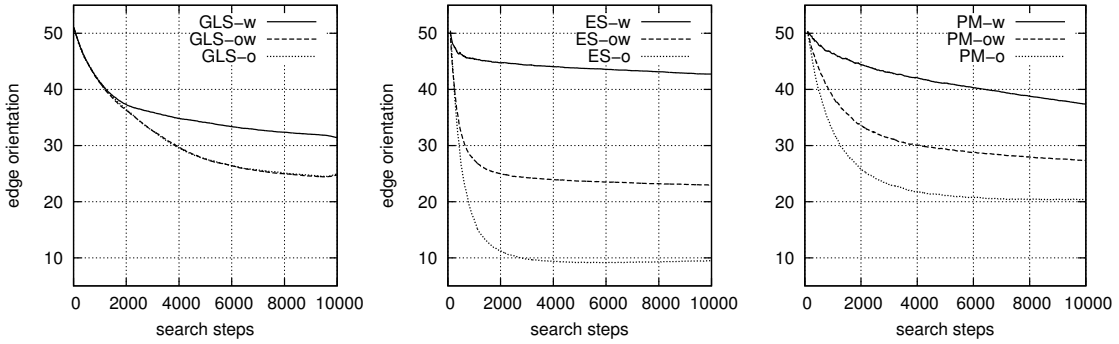
(a) GLS **(b)** ES **(c)** PM

Figure 4.2: Average edge weight over the optimization progress for different configurations and Euclidean instances with $n = 50$

mainly use intensification. Furthermore, the plots indicate a difference between the three ES variants and the three PM variants. EAs that consider orientation and weights perform better than EAs that consider only weights, which, in turn, perform better than those that consider only orientation. In contrast, the progress of the GLS variants is almost identical.

As expected and as seen in the results, ES performance increases if more problem-specific knowledge is integrated. In contrast with GLS, considering additional knowledge has no influence. To further investigate the influence of problem-specific knowledge, we analyze how the average edge weights and edge orientation changes during a run. Figures 4.2 and 4.3 plot the average weight w_{ij} (Figure 4.2) and the average edge orientation γ_{ij} (Figure 4.3) of the edges in the current best found solution over the number of search steps.

For ES, the choice of problem-specific knowledge has a strong influence on the properties of the found solution. When considering only orientation (ES-o), an ES run finds good solutions with many edges that point towards the graph center (low γ; Figure 4.3). Edge weights are also reduced (Figure 4.2), but less so than other variants or than to the change of γ. When considering only edge weights (ES-w), we observe the opposite behavior: ES finds solutions with low-weight edges (Figure 4.2) that do not necessarily point towards the center of the graph (Figure 4.3). If we consider both weights and orientation (ES-ow), we are able to find solutions with low-weight edges that point towards the graph center, however, we find less of these if we consider only edge weights (ES-w) or orientation (ES-o), respectively. In contrast, for GLS, the choice of the cost of solution feature does not have great influence. All three variants (using either w^w_{ij}, w^o_{ij}, or w^{ow}_{ij}) show similar behavior. Keeping in mind that GLS finds better solutions than ES, we see that penalizing



(a) GLS **(b)** ES **(c)** PM

Figure 4.3: Average edge orientation over the optimization progress for different configurations and Euclidean instances with $n = 50$

edges that have large weights or that do not point towards the center of a graph is a more promising strategy than favoring low-weight edges that point towards the tree's center.

Table 4.4 lists the average CPU time t in seconds for one optimization run. All heuristics were implemented in C++ and executed on an Intel Core i5 CPU with 2.67GHz and 4 GBytes of memory running Linux 2.6.38. For small instances the differences are small. For larger instances the EAs using the PM crossover are slower than the rest. This can be explained by the slightly more complicated crossover procedure used. Using different weighting of edges has no significant influence on the CPU times.

4.4.5 Clustered instances

Table 4.5 presents the results for the clustered instances. The experimental setup is identical to the experiment in the previous paragraphs. Again, GLS clearly outperforms ES for larger problem instances. The different GLS configurations do not significantly differ. Also clustered OCST instance have different characteristics than regular Euclidean instances, the results for the analyzed optimization methods are similar. This indicates a robust search performance for both the EA configurations and GLS.

Table 4.4: Runtime comparison for Euclidean OCST instances

n		ES	hES-w	hES-o	hES-ow	PM-w	PM-o	PM-ow	GLS-w	GLS-o	GLS-ow
10	$t[s]$	0.03	0.03	0.03	0.03	0.04	0.04	0.04	0.02	0.02	0.02
	σ	0.0	0.01	0.0	0.01	0.0	0.0	0.0	0.0	0.0	0.0
15	$t[s]$	0.06	0.06	0.06	0.06	0.08	0.08	0.08	0.05	0.05	0.05
	σ	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
25	$t[s]$	0.21	0.21	0.2	0.2	0.32	0.32	0.32	0.2	0.19	0.19
	σ	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
50	$t[s]$	1.72	1.74	1.54	1.61	2.53	2.52	2.5	1.69	1.67	1.67
	σ	0.05	0.05	0.06	0.05	0.05	0.05	0.05	0.09	0.09	0.09
75	$t[s]$	8.77	9.22	7.68	8.35	12.66	12.41	12.43	8.71	8.65	8.63
	σ	0.36	0.34	0.34	0.32	0.36	0.35	0.37	0.53	0.56	0.51
100	$t[s]$	39.86	43.78	36.73	38.81	57.08	55.53	55.56	39.68	39.72	39.34
	σ	1.84	1.55	1.72	1.61	1.82	1.91	1.97	2.38	2.41	2.64

Table 4.5: Performance comparison for clustered instances

n		ES	hES-w	hES-o	hES-ow	PM-w	PM-o	PM-ow	GLS-w	GLS-o	GLS-ow
10	P_{suc}	0.43	0.59	0.09	0.72	0.77	0.45	**0.79**	0.61	0.54	0.55
	$\mu(f(T))$	1834.75	1829.12	1899.11	1824.97	1824.32	1833.79	**1823.62**	1833.43	1836.1	1835.59
	σ	15.28	6.0	31.96	3.36	5.39	13.82	4.74	17.53	20.1	19.63
15	$\mu(f(T))$	4470.46	4347.9	4817.63	4309.07	4314.32	4400.54	**4304.19**	4336.3	4353.07	4353.04
	σ	93.54	29.3	135.14	16.04	29.48	61.16	24.54	50.67	58.7	58.57
25	$\mu(f(T))$	10431	9391.36	11918	9191.66	9223.15	9742.19	9187.07	**9170.2**	9266.98	9268.32
	σ	416.33	103.47	591.27	61.46	90.31	216.39	80.37	98.26	127.89	130.66
50	$\mu(f(T))$	40088	29892	48378	29026	28894	33035	28778	**28244**	28625	28612
	σ	3382.35	450.94	4108.01	321.19	286.19	1180.75	276.28	251.97	387.18	385.52
75	$\mu(f(T))$	75979	59069	95541	57748	56783	65010	56659	**55635**	56140	56108
	σ	9216.58	786.65	8434.85	639.95	444.14	2530.43	440.97	485.22	630.54	642.62
100	$\mu(f(T))$	92573	86168	121346	85057	83131	90158	83075	**81979**	82425	82445
	σ	4427.71	951.9	8704.36	745.48	479.35	2002.24	494.69	607.36	660.07	683.07

4.5 Conclusions

It is known that edges in optimal solutions of OCST problems have, on average, low distance weights and point towards the center of a tree. Consequently, we can make use of this knowledge for the initial solution, for building redundant representations that prefer solutions similar to optimal solutions, for search operators, and, as it was demonstrated here, for the design of fitness functions.

This work demonstrates how the performance of heuristic optimization methods can be increased by considering the properties of optimal solutions for the fitness function. In contrast to other approaches which reward promising solution features by favoring edges with low weight pointing towards the tree's center, we present a guided local search approach which penalizes low-quality edges with large weights that do not point towards the tree's center. Experiments show that GLS outperforms EAs using state-of-the-art search operators for larger problem instances. However, we are not able to modify or increase GLS performance when we penalize either low weight edges, edges with wrong orientation, or both. Instead, GLS performance is independent of the type of penalization. This is in contrast to the classical assumption that considering more problem-specific knowledge about high-quality solutions increases search performance.

Future work will analyze how GLS can be modified to reward high-quality solution features instead of adding penalties to low-quality solutions. This would allow us to study in greater detail why punishing low-quality features is more successful than rewarding high-quality solution features. Another interesting topic is to change the evaluation function in other types of metaheuristics.

References

Ahuja, R. K. and V. V. S. Murty (1987a). "Exact and heuristic algorithms for the optimum communication spanning tree problem". In: *Transportation Science* 21.3, pp. 163–170.

Carrano, E. G., R. H. C. Takahashi, C. M. Fonseca, and O. M. Neto (2010). "Non-linear network optimization: an embedding vector space approach". In: *IEEE Transaction on Evolutionary Computation* 14 (2), pp. 206–226.

Contreras, I., E. Fernández, and A. Marín (2010). "Lagrangean bounds for the optimum communication spanning tree problem". In: *TOP* 18 (1). 10.1007/s11750-009-0112-5, pp. 140–157.

Fischer, T. and P. Merz (2007). "A memetic algorithm for the optimum communication spanning tree problem". In: *Hybrid Metaheuristics*. Ed. by T. Bartz-Beielstein, M. J. B. Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels. Vol. 4771. Springer, pp. 170–184.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Hu, T. C. (1974). "Optimum communication spanning trees." In: *SIAM Journal on Computing* 3.3, pp. 188–195.

Palmer, C. and A. Kershenbaum (1995). "An approach to a problem in network design using genetic algorithms". In: *Networks* 26, pp. 151–163.

Papadimitriou, C. H. and M. Yannakakis (1991). "Optimization, approximation, and complexity classes". In: *Journal of Computer and System Sciences* 43.3, pp. 425–440.

Peleg, D. and E. Reshef (1998). "Deterministic polylog approximation for minimum communication spanning trees". In: *ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming*. London, UK: Springer-Verlag, pp. 670–681.

Raidl, G. R. and B. A. Julstrom (2003a). "Edge sets: an effective evolutionary coding of spanning trees". In: *IEEE Transactions on Evolutionary Computation* 7.3, pp. 225–239.

Rothlauf, F. (2009a). "An encoding in metaheuristics for the minimum communication spanning tree problem." In: *INFORMS Journal on Computing* 21.4, pp. 575–584.

Rothlauf, F. (2009b). "On Optimal Solutions for the Optimal Communication Spanning Tree Problem". In: *Operations Research* 57.2, pp. 413–425.

Rothlauf, F. (2009c). "On the Bias and Performance of the Edge-Set Encoding". In: *IEEE Transactions on Evolutionary Computation* 13.3, pp. 486–499.

Sharma, P. (2006). "Algorithms for the optimum communication spanning tree problem". In: *Annals of Operations Research* 143.1, pp. 203–209.

Soak, S.-M. (2006). "A new evolutionary approach for the optimal communication spanning tree problem". In: *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences* E89-A.10, pp. 2882–2893.

Steitz, W. and F. Rothlauf (2008). "Orientation matters: how to efficiently solve OCST problems with problem-specific EAs". In: *GECCO '08: Proceedings of*

the 10th annual conference on Genetic and evolutionary computation, Atlanta, GA, USA. New York, NY, USA: ACM, pp. 563–570.

Tsang, E., P. Mills, and J. Ford (2002). *Extending Guided Local Search - Towards a Metaheuristic Algorithm With No Parameters To Tune*. Tech. rep. 371. Department of Computer Science, University of Essex.

Voudouris, C. (1997). "Guided Local search for combinatorial optimization problems". PhD thesis. Colchester,UK: Department of computer science, University of Essex.

Voudouris, C. (2003). "Guided local sarch". In: *Handbook of Metaheuristics*. Ed. by F. Glover and G. A. Kochenberger. Springer, pp. 185–218.

Voudouris, C. and E. Tsang (1999). "Guided local search and its application to the traveling salesman problem". In: *European Journal of Operational Research* 113.2, pp. 469–499.

Wu, B. Y., K.-M. Chao, and C. Y. Tang (2000a). "A polynomial time approximation scheme for optimal product-requirement communication spanning trees". In: *Journal of Algorithms* 36.2, pp. 182–204.

Chapter 5

Biased metaheuristics for the quadratic minimum spanning tree problem

Wolfgang Steitz, Franz Rothlauf

Abstract

The quadratic minimum spanning tree (QMST) problem is an extension of the classical minimum spanning tree problem. Besides the edge costs, so called intercosts arise through the interplay of each pair of edges in the tree. Edge costs and intercosts are linked additively. Since the problem is difficult to solve, researchers came up with a wide range of heuristic optimization approaches. However, all of these works ignore the fact that the height of the edge costs versus the height of the intercosts considerably influences the structure of optimal solutions and therefore the difficulty of the problem. In this paper, we design construction heuristics that take into account the ratio between edge costs and intercosts. These insights are then used to create mutation and crossover operators in a similar way. Using these operators in simulated annealing and an evolutionary algorithm allows us to significantly improve upon the state-of-the-art.

5.1 Introduction

The $\mathcal{NP}$-hard quadratic minimum spanning tree (QMST) problem (Assad and Xu, 1992) is a combinatorial optimization problem that seeks a spanning tree with minimal total costs. In contrast to the classical minimum spanning tree problem, it considers edge weights as well as so called intercosts, which are associated with each pair of edges. There are practical applications for QMST problems, where intercosts represent additional effort for connecting different types of oil pipelines. In a similar way, intercosts can represent the costs of turns in route planning (Geisberger and Vetter, 2011; Winter, 2002).

The current state-of-the-art approaches mainly use population-based metaheuristics (Palubeckis, Rubliauskas, and Targamadze, 2010; Soak, Corne, and Ahn, 2006; Sundar and Singh, 2010). The performance of these approaches is evaluated using randomly created instances with different problem sizes. With larger problem size, problem difficulty increases. A second parameter that influences problem difficulty is the ratio between edge weights and intercosts. Instances with high edge weights and low intercosts, are easier to solve, since optimal solutions are similar to the minimum spanning tree. In the existing literature, the influence of different ratios is ignored.

Metaheuristics are high-level optimization methods, that are applied to a wide range of combinatorial optimization problems. While their application range is huge, the search performance of out-of-the-box metaheuristics is often poor. One way to design effective and efficient metaheuristics is to incorporate problem-specific knowledge into the algorithm (Bonissone et al., 2006; Droste and Wiesmann, 2003; Hauschild et al., 2012; Raidl and Julstrom, 2003a; Rothlauf, 2011). This results from the no-free-lunch theorem (Wolpert and Macready, 1997) as well as experience from practical applications. Problem-specific knowledge may be used to bias the individual components of a metaheuristics (Rothlauf, 2011). Indeed, one can:

1. Initialize the metaheuristic with solutions that have similar properties to high-quality solutions.
2. Use a redundant representation that over-represents solutions with certain properties.
3. Use search operators that favor building blocks, similar to those of known good solutions.
4. Assign higher fitness values to solutions with similar characteristics to high-quality solutions.

In order to make use of problem-specific knowledge in a metaheuristic, a thorough problem understanding is mandatory, since introducing a wrong bias would hinder

successful optimization. Problem-specific knowledge can be gathered by comparing the structure of optimal and random solutions for small, solvable problem instances (Rothlauf, 2009c; Steitz and Rothlauf, 2012). In this paper, we choose a different approach and generate solutions with different biased construction heuristics. We compare the resulting performance and compare the found solutions. Finally, we apply the findings to the design of heuristic search operators (e.g. mutation and recombination).

For the QMST problem, no analysis of the problem structure exists. Existing heuristics often use some approaches tailored to the problem, but these are only based on assumptions and do not necessarily lead to high-performing algorithms. Therefore the goal of this paper is to analyze the QMST problem and to build biased search operators based on this problem analysis. These operators are compared with the current state-of-the-art approaches. In summary, the main findings of this work are:

1. The performance of heuristics for the QMST problem strongly depends on the ratio between edge costs and intercosts. Successful construction heuristics, algorithms that create solutions by iteratively appending edges to a partial solution until a complete spanning tree is created, are biased towards low-weight edges that have low intercosts with the current partial solution. These solutions share many edges with optimal solutions and have lower costs than trees created using other approaches. The search performance of metaheuristics is improved, when starting from a solution created by our construction heuristic.

2. Based on the insights gained from developing and analyzing construction heuristics, we propose biased search operators for the QMST problem. These search operators outperform existing operators proposed in the literature. Using these heuristic operators, we are able to significantly outperform the current state-of-the-art approaches.

Section 5.2 defines the QMST problem and gives an overview of existing solution approaches. In Section 5.3, we present and analyze construction heuristics. Sections 5.4 and 5.5 present and study biased and problem-specific local search and recombination operators, resp. Section 6.5 presents a comparison to the current state-of-the-art approaches. The paper ends with concluding remarks.

5.2 The Quadratic Minimum Spanning Tree Problem

5.2.1 Problem formulation

The quadratic minimum spanning tree problem (QMST) is a combinatorial tree optimization problem, first described by Assad and Xu (1992). Similarly to other spanning tree problems, costs are associated with each edge. Additionally, there are intercosts that arise for each pair of edges included in a solution. Given a collection of nodes, we seek a spanning tree that connects all the nodes and leads to minimum total cost, defined to be the sum of the cost associated with each edge and the intercost of each pair of edges in the spanning tree.

Formally, let $G = (V, E)$ be a complete, weighted, undirected graph with $V = \{v_1 \dots v_n\}$ nodes and $E = \{e_1 \dots e_m\}$ edges. The $n \times n$ *distance matrix* $C = c(e)$ defines the weight or cost $c(e)$ of an edge e; the $m \times m$ *intercost matrix* $Q = q(e, f)$ defines the cost associated with each pairs of edges. In the Euclidean case, the distance weights $c(e)$ are the Euclidean distances between the nodes. The QMST problem seeks a tree $T = (V, F)$ with $F \subseteq E$ and $|F| = n - 1$ which minimizes

$$f(T) = w_e \sum_{e \in F} c(e) + w_i \sum_{e_1 \in F} \sum_{e_2 \in F} q(e_1, e_2), \tag{5.1}$$

where the coefficients w_e and w_i weight the edge weights and the intercosts, resp. The distance $d(T_1, T_2)$ between two spanning trees T_1 and T_2 is the number of edges that differ. It can be calculated by subtracting the number of common edges from the number of edges in one tree $d(T_1, T_2) = |E_1| - |E_1 \cap E_2|$.

The QMST problem is proven $\mathcal{NP}$-hard by Assad and Xu (1992). The authors also proposed a branch-and-bound approach and two heuristics for the QMST problem. They presented experimental results for small instances of size $n \leq 15$. Zhou and Gen (1998) proposed a genetic algorithm based on Prüfer number encoding, which outperforms the heuristics of Assad and Xu (1992). Soak, Corne, and Ahn (2006) proposed the edge-window-decoder (EWD) encoding and analyzed the performance of evolutionary algorithms using different types of encodings (EWD, edge-sets encoding, NetKey encoding, Prüfer encoding, and node-biased encoding). EWD outperformed the other encoding, with a heuristic version of the edge-set encoding (Raidl and Julstrom, 2003a) being the second best. They used a rather small test set with only six Euclidean instances.

Öncan and Punnen (2010) developed a Lagrangian relaxation procedure, which generates tight lower bounds. They also present a tabu search heuristic, whose performance is similar to the genetic algorithm by Zhou and Gen (1998).

Palubeckis, Rubliauskas, and Targamadze (2010) presented simulated annealing, another genetic algorithm, and an iterated tabu search approach for the QMST. In their experiments, the tabu search approach performed best of the three approaches. Unfortunately, they did not compare their approaches to the others. Sundar and Singh (2010) present a swarm intelligence approach based on the artificial bee colony (ABC) metaheuristic from Karaboga and Basturk (2007). ABC uses a population of solutions, which is improved by mutating solutions and creating new solutions as a combination of several solutions. Sundar and Singh (2010) also developed heuristic operators for the QMST problem. Solutions are represented using the edge set encoding (Raidl and Julstrom, 2003a). The initial population of solutions is created by using a specific initialization method (for details see Sect. 5.3). The authors compare their approach to the existing evolutionary approaches of Zhou and Gen (1998) and Soak, Corne, and Ahn (2006). ABC outperforms these two, but uses more computation time.

5.2.2 Creating test instances

For the experiments in this paper, we follow previous work (Öncan and Punnen, 2010; Sundar and Singh, 2010) and use randomly created QMST instances. The real-valued intercosts and distance costs c_e are either uniformly distributed in $]0, 1]$, or the nodes are randomly placed in the unit square and the c_e are the Euclidean distances.

We study instances with different ratios of $r = w_e/w_i$. Previous publications only used constant ratios r. However, this creates only certain types of instances. For example, high values of r lead to instances, where optimal solutions are similar to minimal spanning trees. This often makes solving these instances easy as many solution approaches prefer solutions similar to an MST. In the extreme case of $r \to \infty$, the QMST problem becomes the MST problem. To get a better understanding of QMST instances, we create instances of different size n with different setting of w_e, w_i.

5.3 Construction heuristics

Construction heuristics (also known as single-pass heuristics) usually build a solution from scratch in a step-wise creation process where in each step parts of the solution are fixed (**Fisher1988**). The created solutions can be used to seed a metaheuristic, what often leads to better solutions and lower running times. This section proposes construction heuristics for the QMST problem and analyzes the resulting performance of metaheuristics using different initial solutions.

In (meta)heuristics for spanning tree problems, the minimal spanning tree (MST) is often used as initial solution. This is due to the low effort for creating MSTs ($O(n^2)$ using Prim's algorithm) and the fact that the objective function often directly depends on the edge weights. Prominent examples are the TSP or Steiner tree problem. For the TSP, the 3/2-approximation algorithm from Christofides (1976) starts with an MST, computes a minimum cost perfect matching on the set of odd-degree nodes, and then generates an Eulerian graph. For the Steiner tree problem, Robins and Zelikovsky (2005) presented a polynomial-time approximation scheme that starts with an MST and results in the best-known performance ratio approaching $1 + \ln 3/2 \approx 1.55$. Hwang (1976) showed for the rectilinear Steiner tree problem that an MST itself is a 3/2-approximation. Consequently, a variety of approaches either start with an MST and improve this solution in subsequent steps (T.-H. Chao and Hsu, 1994; Ho, Vijayan, and Wong, 1990; Julstrom and Raidl, 2002; Robins and Zelikovsky, 2005), or imitate the MST construction of Kruskal and Prim (Bern, 1988; Richards, 1989).

Sundar and Singh (2010) suggested a construction heuristic for the QMST problem. The purpose of the construction heuristic is to create high-quality initial solutions for a subsequent metaheuristic. The approach assigns a "potential cost" $c'_p(e_l) = w_e c(e_l) + w_i \sum_{m \neq l} q(e_l, e_m)$ to each edge e_l. Solutions are iteratively created by an approach resembling Prim's algorithm. In each iteration, the approach selects an edge using roulette wheel selection. Thus, the probability of selecting an edge e is inversely proportional to its potential cost $c'_p(e)$. If the selected edge does not create a cycle, it is added to the partial solution. The algorithm stops after the insertion of $n-1$ edges. A drawback of this approach is that c'_p over-emphasizes the influence of intercosts. Although only $n-2$ intercost values are relevant for $f(T)$, c_p sums up $\frac{n(n-1)}{2} - 1$ intercost values. Consequently, c'_p should be normalized. We suggest to normalize the influence of the intercosts by the factor $(n-2)/(\frac{n(n-1)}{2} - 1)$ leading to the normalized potential cost

$$c_p(e_l) = w_e c(e_l) + \frac{2w_i}{n+1} \sum_{m \neq l} q(e_l, e_m). \tag{5.2}$$

Construction heuristics using the (normalized) potential cost are fast as the potential cost values can be calculated a priori (time effort $O(n^4)$) and stored in an appropriate data structure. However, the principle of potential cost is not meaningful. The potential cost of an edge e_l depends on its weight $c(e_l)$ and the average intercost $\frac{1}{n-2} \sum q(e_l, e_m)$ between e_l and all other edges $e_m \neq e_l$. However, if all intercost values are drawn from the same distribution, the expected mean of $\sum q(e_l, e_m)$ is equal for all subsamples of size $n-2$. As a result, the potential cost values are determined only by the distance weights and not by the intercost which only add a constant factor (if all intercost values follow the same distribution).

A more promising way to construct an initial solution for the QMST problem is to consider only the intercosts between an edge e_l and those edges e_m that are already part of the partial tree T_l. Consequently, the *dynamic cost* c_d of an edge e_l depends on only those edges that have already been added to the partial tree T_l in previous steps of the construction heuristic:

$$c_d(e_l, T_l) = w_e c(e_l) + w_i \sum_{e_m \in T_l} q(e_l, e_m) \tag{5.3}$$

A construction heuristic using dynamic cost considers intercosts more exactly. However, the intercost value of an edge depend on the partial tree T_i, which makes the a-priori calculation of dynamic cost values infeasible. Instead, dynamic cost values are updated in each iteration leading to a higher effort for constructing a tree.

We suggest constructing spanning trees using Kruskal's algorithm and dynamic costs. In each iteration l, the edge $e_l \neq T_l$ with lowest weight $c_d(e_l)$ is added to the partial tree T_l, as long as it does not create a cycle. The approach starts with an empty partial tree T_0. The construction heuristic is deterministic and always creates the same tree for given distance weights and intercosts.

To compare the performance of the different construction heuristics, we create random QMST instances of different sizes $n \in \{25, 50, 75, 100\}$ (see Sect. 5.2.2). For each size, we create 100 random and Euclidean problem instances and consider three different ratios $r \in \{0.2, 1.0, 5\}$, which lead to three considerably different types of instances. We compare four construction heuristics:

- *Roulette* c'_p*:* The construction heuristic proposed by Sundar and Singh (2010) with potential cost c'_p and roulette wheel selection of edges.
- *Kruskal* c'_p*:* Kruskal's algorithm using potential cost c'_p
- *Kruskal* c_p*:* Kruskal's algorithm using normalized potential cost c_p
- *Kruskal* c_d*:* Kruskal's algorithm using dynamic cost c_d

Tables 5.1 (Euclidean instances) and 5.2 (random instances) show the cost f of the generated trees averaged over all problem instances. For comparison, we present the average cost of a random tree and an MST. For the non-deterministic tree construction approaches (Random and Roulette c'_p), we generate 100 trees for each problem instances. Standard deviations are in brackets.

Kruskal's algorithm using dynamic costs c_d performs significantly better than the other construction heuristics for all different problem types. Differences are significant using a ranked t-test with an error level of $p < 0.01$. In comparison to the second best construction heuristic (Kruskal c_p), the average cost is lower up to

Table 5.1: Performance of construction heuristics for Euclidean instances

n	r	Random		MST	Roulette c'_p		Kruskal c'_p	Kruskal c_p	Kruskal c_d
25	0.2	139,533	(4,595)	138,451	138,834	(4,576)	122,549	122,611	**92,459**
50	0.2	590,412	(9,705)	588,795	590,520	(9,632)	550,402	550,360	**432,328**
75	0.2	1,355,023	(14,301)	1,350,272	1,354,716	(14,507)	1,285,088	1,283,764	**1,038,690**
100	0.2	2,430,521	(20,304)	2,426,832	2,430,406	(19,740)	2,339,136	2,339,604	**1,918,825**
25	1.0	28,842	(928)	27,943	28,760	(923)	25,662	25,522	**19,433**
50	1.0	120,150	(1,928)	118,235	120,215	(2,017)	111,747	111,467	**88,147**
75	1.0	273,805	(2,986)	271,134	273,673	(3,048)	260,550	260,830	**210,498**
100	1.0	490,392	(4,033)	486,136	490,290	(3,951)	471,458	469,994	**387,228**
25	5.0	33,772	(1,133)	29,333	33,707	(1,080)	29,423	28,083	**22,999**
50	5.0	130,449	(2,083)	119,844	130,315	(2,134)	120,665	116,246	**95,565**
75	5.0	289,477	(3,046)	273,125	289,327	(2,957)	274,680	266,958	**222,185**
100	5.0	510,960	(4,096)	487,790	511,016	(4,059)	490,279	479,504	**403,041**

Table 5.2: Performance of construction heuristics for random instances

n	r	Random		MST	Roulette c'_p		Kruskal c'_p	Kruskal c_p	Kruskal c_d
25	0.2	139,000	(4,697)	137,950	138,669	(4,657)	122,609	122,262	**92,415**
50	0.2	590,786	(9,571)	588,776	589,987	(9,670)	551,494	550,301	**432,519**
75	0.2	1,354,118	(14,697)	1,352,183	1,354,243	(14,622)	1,288,516	1,288,841	**1,037,584**
100	0.2	2,431,348	(19,587)	2,426,159	2,429,686	(19,073)	2,336,747	2,335,688	**1,921,331**
25	1.0	28,891	(998)	27,695	28,731	(965)	25,569	25,351	**19,441**
50	1.0	120,024	(1,941)	117,776	120,051	(1,944)	111,633	111,317	**87,901**
75	1.0	273,788	(2,992)	269,804	273,563	(2,869)	260,316	259,476	**210,175**
100	1.0	490,204	(4,073)	485,001	489,937	(3,997)	471,037	469,745	**386,548**
25	5.0	33,579	(1,138)	28,326	33,531	(1,110)	28,649	26,961	**22,152**
50	5.0	129,844	(2,149)	117,899	129,818	(2,165)	119,558	114,153	**94,068**
75	5.0	288,627	(3,159)	270,330	288,691	(3,093)	272,686	263,462	**219,744**
100	5.0	509,556	(4,146)	485,660	510,112	(4,385)	487,371	474,377	**399,694**

Table 5.3: Runtimes of construction heuristics

n	Random	MST	Roulette c'_p	Kruskal c'_p	Kruskal c_p	Kruskal c_d
25	0.00	0.00	0.00	0.00	0.00	0.00
50	0.00	0.00	0.01	0.00	0.00	0.01
75	0.00	0.00	0.05	0.03	0.03	0.04
100	0.00	0.00	0.12	0.07	0.07	0.10

25%. For higher values of r ($r = 5$), an MST has lower cost than a randomly created tree (significant, $p < 0.01$), as it is relevant to construct trees using edges with low edge weights. For low values of r ($r = 0.2$), it is more important to create trees with low intercost. Thus, the average cost of an MST is not significantly different from random solutions. The construction heuristic of (Sundar and Singh, 2010) creates solutions with cost that are not significantly different from random trees. Thus, this approach is not able to create high-quality solutions. Combining the potential cost with Kruskal's algorithm, leads to better solutions in comparison to the original approach using roulette wheel selection (significant, $p < 0.1$). Normalizing potential cost (Kruskal c_p) does not lead to significantly better solutions than Kruskal c'_p for low values of r ($r = 0.2$ and $r = 1$), but to a significant improvement for $r = 5$ (significant, $p = 0.01$)). As optimal solutions for problems with higher r are more MST-like, reducing the weight of intercost allows Kruskal's algorithm to construct better and more MST-like trees.

Comparing the runtimes of the different construction heuristics in Table 5.3 shows some significant differences. Kruskal's algorithm using c'_p and c_p is rather fast. But the calculation of potential costs takes some time, and therefore creating a MST is faster. Using dynamic costs increases the runtimes slightly. Roulette wheel selection is the slowest construction heuristic.

5.3.1 Biased initialization

Metaheuristics that use solutions created by a construction heuristic as an initial solutions often have a head start and return better solutions. We study the performance of an evolutionary algorithm (EA) and a simulated annealing (SA) approach as representative examples of local and recombination-based search using different construction heuristics for generating initial solutions.

The EA is based on the approach of Soak, Corne, and Ahn (2006), which combines a standard steady-state EA with the edge-set encoding of (Raidl and Julstrom, 2003a). The mutation operator (denoted by Raidl and Julstrom (2003a) as "insertion before deletion") randomly replaces one edge in a tree by first including a random

edge not in the tree and then removing a random edge from the created cycle. The crossover operator (denoted by Raidl and Julstrom (2003a) as "Kruskal*") selects two random solutions T_1 and T_2. In a first step, it includes all edges $(E_1 \cap E_2)$ in the offspring T_{off}. Then, T_{off} is completed by applying a variant of Kruskal's algorithm that ignores the edge weights to the remaining edges $(E_1 \cup E_2) \backslash (E_1 \cap E_2)$. In each iteration of the EA, two solutions are sampled randomly from the population and crossover is performed with probability $p_c = 1$. The offspring T_{off} is then mutated exactly once (probability $p_m = 1$). T_{off} replaces the worst individual in the population if the fitness is better $c(T_{off}) \leq \max(c(T_i))$ for $i \in \{0, \ldots, N-1\}$). For the experiments, we use a population size of $N = 100$.

Simulated annealing (SA) (Kirkpatrick, Gelatt, and Vecchi, 1983) creates a neighboring solution T_{off} in every iteration by applying exactly one mutation to a parent solution T_{par}. If there is an improvement, $c(T_{off}) < c(T_{par})$, T_{off} replaces T_{par}. If $c(T_{off}) \geq c(T_{par})$, T_{par} is replaced with probability $P(U) = exp(-(c(T_{off}) - c(T_{par}))/U)$. With lower temperature U, the probability of accepting worse solutions decreases. We use the same insertion-before-deletion operator as for the EA and set the initial temperature U_0 with respect to the test instance at hand. For each instance, we create 1,000 random solutions and set the initial temperature to $U_0 = 0.1 \times \sigma(c(T_i))$, where $\sigma(c(T_i))$ is the standard deviation of the cost of a randomly created solution T_i. In each iteration, the temperature is reduced by a constant factor, $U_{t+1} = 0.99 \times U_t$.

We compare the performance of the metaheuristics for Euclidean instances with different r. For each setting, we randomly create 100 problem instances. Results for random problem instances are omitted as they are analogous to Euclidean instances. For different initial solutions T_s, Table 5.4 lists the average and standard deviation (in brackets) of the cost of the best found solution for EA and SA. The initial solution is either chosen randomly, an MST, or generated by the construction heuristics Kruskal c_p or Kruskal c_d. The SA starts with T_s. For the population-based GA, one randomly chosen individual in the initial population of size $N = 100$ is set to T_s. All other solutions in the initial population are random trees. We perform 10 independent runs for each problem instance; each run is terminated after $200n$ evaluations.

For both types of metaheuristics EA and SA, starting with solutions generated by Kruskal's algorithm using dynamic cost c_d leads to better performance than starting with a different initial solution (significant for $n > 25$, $p < 0.01$). Using Kruskal's algorithm with normalized potential cost c_p allows the metaheuristics to find better solutions in comparison to starting with an MST or a random solution (also significant for $n > 25$ with $p < 0.01$).

The performance differences between starting with different initial solutions T_i are analogous for the EA and SA, which confirms that the differences are a result

of different T_i and the performance of metaheuristics can be improved by initializing them with solutions created by appropriate construction heuristics. For the two representative metaheuristics, SA outperforms the EA approach. This can be explained by the stronger intensification leading to a more focused search and the stronger bias of the SA (for the EA, only one out of 100 initial solutions is selected to be T_i). In principle, the performance of the metaheuristics can be easily increased by playing around with the different parameters of the metaheuristics. However, this is not the purpose of this study; we study whether biasing the initial solutions influences the performance of representative examples of metaheuristics. We believe that an appropriate bias of initial solutions, which considers the characteristics of the problem at hand, is one possibility for designing successful metaheuristics.

5.4 Biased local search operators

We analyze how local search operators can be improved by including problem-specific knowledge. We use the SA from Sect. 5.3.1 as representative example of a metaheuristic based on iterated local search steps and study different different construction heuristics defining the functionality of mutation.

The SA uses two types of mutation operators to create new solutions, a *heuristic* and a *non-heuristic* variant. Both are based on the mutation operator of the edge-set encoding presented by Raidl and Julstrom (2003a) and used for the QMST problem by Soak, Corne, and Ahn (2006). The non-heuristic variant replaces a random edge in the spanning tree T using the "insertion before deletion" operator. The heuristic variant does not insert an edge randomly, but according to either its weight c_e, potential cost c_p, or dynamic cost c_d. The edge to be inserted is selected via tournament selection. In the dynamic case, this allows to compute the costs only for a subset of the available edges. The tournament size is twenty. The edge with lowest weight in the tournament is inserted in T. Finally, a random edge of the created cycle is removed from the solution.

We study the performance of an SA for instances of different problem sizes $n = \{25, 50\ , 75, 100\}$. SA parameters are set as described in Sect. 5.3.1. For each problem instance, we randomly create 10 Euclidean instances with $r \in \{0.2, 1, 5\}$. We perform 10 SA runs (with different seeds) for each problem instance. The SA starts with a random tree T_s.

Table 5.5 shows the mean and standard deviation (in brackets) of the cost f of the best solution found during a SA run. The results are for the non-heuristic variant (rand) and for the heuristic variant sorting the edges either according to the distance weights (c_e), normalized potential cost (c_p), or dynamic cost (c_d). Table 5.6 lists

Table 5.4: Performance comparison biased initialization

n	r	SA *Random*	SA *MST*	SA c_p	SA c_d	EA *Random*	EA *MST*	EA c_p	EA c_d
25	0.2	93,233	93,114	92,066	**92,054**	96,312	96,418	95,340	92,716
		(2,189)	(2,193)	(2,047)	(683)	(2,214)	(2,383)	(2,060)	(416)
50	0.2	443,742	444,040	440,338	**431,783**	457,254	457,294	452,441	432,671
		(4,539)	(4,575)	(4,410)	(1,131)	(4,944)	(4,940)	(4,663)	(676)
75	0.2	1,073,279	1,073,150	1,067,472	**1,038,320**	1,101,619	1,101,590	1,091,914	1,038,909
		(6,960)	(6,668)	(6,717)	(1,355)	(7,712)	(7,620)	(7,374)	(908)
100	0.2	1,989,968	1,989,068	1,981,305	**1,918,561**	2,034,906	2,033,781	2,019,469	1,918,812
		(9,455)	(9,425)	(8,884)	(1,936)	(10,427)	(10,228)	(9,838)	(1,208)
25	1.0	19,522	19,452	**19,262**	19,370	20,169	20,118	19,862	19,509
		(437)	(417)	(403)	(132)	(460)	(477)	(435)	(90.53)
50	1.0	90,461	90,338	89,668	**87,951**	93,209	93,123	92,058	88,138
		(920)	(902)	(857)	(238)	(1,002)	(1,001)	(951)	(138)
75	1.0	217,240	217,085	215,977	**210,007**	222,828	222,702	220,689	210,141
		(1,457)	(1,400)	(1,353)	(280)	(1,506)	(1,541)	(1,481)	(187)
100	1.0	401,307	401,008	399,313	**386,806**	410,258	410,188	407,034	386,841
		(1,868)	(1,851)	(1,869)	(348)	(2,127)	(2,112)	(1,996)	(237)
25	5.0	23,068	22,874	22,756	**22,695**	23,527	23,369	23,197	22,841
		(509)	(432)	(404)	(136)	(458)	(411)	(365)	(85.06)
50	5.0	98,400	97,595	97,210	**95,378**	100,755	99,910	99,138	95,581
		(1,007)	(951)	(896)	(240)	(1,082)	(961)	(886)	(132)
75	5.0	229,758	228,354	227,393	**221,609**	235,007	232,992	231,595	221,732
		(1,540)	(1,402)	(1,402)	(281)	(1,575)	(1,525)	(1,429)	(185)
100	5.0	418,357	416,406	414,974	**402,581**	427,110	424,066	421,645	402,619
		(2,033)	(1,867)	(1,898)	(361)	(2,264)	(2,090)	(1,923)	(223)

Table 5.5: Performance comparison of mutation operators

n	r	$rand$	c_e	c_p	c_d
25	0.2	93,266 (2,125)	104,626 (2,450)	96,710 (1,886)	**91,564** (2,217)
50	0.2	443,934 (4,748)	473,320 (4,939)	452,028 (4,463)	**430,529** (4,795)
75	0.2	1,073,490 (6,932)	1,120,706 (7,503)	1,085,576 (7,120)	**1,037,917** (7,073)
100	0.2	1,990,734 (9,493)	2,054,732 (10,311)	2,004,940 (10,003)	**1,924,195** (9,571)
25	1.0	19,562 (455)	21,516 (477)	20,183 (367)	**19,241** (448)
50	1.0	90,564 (912)	95,334 (941)	92,033 (891)	**87,800** (989)
75	1.0	217,300 (1,454)	225,124 (1,455)	219,417 (1,396)	**210,104** (1,457)
100	1.0	401,310 (1,867)	411,863 (2,065)	404,187 (1,985)	**387,914** (1,818)
25	5.0	23,103 (502)	23,781 (435)	23,291 (376)	**22,736** (478)
50	5.0	98,356 (1,012)	99,433 (924)	98,170 (901)	**95,472** (1,081)
75	5.0	229,662 (1,478)	230,657 (1,443)	228,443 (1,429)	**221,925** (1,532)
100	5.0	418,177 (2,017)	418,942 (2,059)	415,864 (1,987)	**404,443** (2,020)

Table 5.6: Runtimes of mutation operators

n	$rand$	c_e	c_p	c_d
25	0.10	0.12	0.12	0.17
50	0.60	0.60	0.62	0.83
75	1.73	1.73	1.77	2.25
100	3.80	3.78	3.88	4.76

the average time (in seconds) of one optimization run using an Intel Core i5 Linux machine with 4GB memory.

The SA configuration using dynamic costs c_d clearly outperforms the other configurations, independently of the problem size n and the ratio r (results are significant using a ranked t-test, $p < 0.001$). Only considering the edge weights (c_d) or using normalized potential costs (c_p) is not better than using no bias at all ($rand$). Only for $r = 5$ SA c_p is able to outperform the non-heuristic variant. Regarding the CPU times, all configurations are fast, but using dynamic costs increases the CPU time about 30%.

5.5 Biased Recombination

Soak, Corne, and Ahn (2006) presented an EA for the QMST problem based on the edge-set encoding (Raidl and Julstrom, 2003a). We modify the recombination operator such that it prefers low-weight/low-cost edges. The recombination operator

creates one offspring tree $T_{off}(E_{off}, V)$ from two parental solutions $T_{p1}(E_{p1}, V)$ and $T_{p2}(E_{p2}, V)$. The operator is designed in such a way that edges with low weight/cost are preferred.

The operator starts with an empty offspring tree T_{off}. In a first step, all edges $E_{p1} \cap E_{p2}$ that exist in both parents are transfered parents are transfered to T_{off}. This preserves substructures appearing in both parents. The offspring is completed by iteratively selecting and adding edges from the remaining parental edges $F_r = (E_{p1} \cup E_{p2}) \setminus (E_{p1} \cap E_{p2})$. Edges that would introduce a cycle, are dropped. The operator stops if the offspring solution T_{off} is completed.

There are different possibilities for selecting edges from F_r (Raidl and Julstrom, 2003a):

- *random selection*: Selects a random edge from F_r.
- *greedy selection*: Selects an edge from F_r with lowest weight/cost. Possible weights/cost are distance weights c_e, potential cost c_p, or dynamic cost c_d.
- *tournament selection*: Selects two random edges from F_r and inserts the one with lower weight/cost into T_{off}.

Greedy and tournament selection are biased towards low-weight/low-cost edges. Based on Raidl and Julstrom (2003a), Soak, Corne, and Ahn (2006) used random selection and greedy selection with respect to potential cost for the QMST problem.

We study the performance of recombination operators using different selection schemes for the EA proposed by Soak, Corne, and Ahn (2006). The steady-state approach iteratively selects two solutions from a population. Crossover combines these two solutions to create a new offspring. One mutation step ("insertion-before-deletion") is applied to each offspring exactly once. The resulting solution replaces the worst solution in the population, if the costs are lower than or equal to the cost of the worst individual in the population $f(T_{off}) \leq \max(f(T_i))$ for $i \in \{0, \ldots, N-1\}$).

Table 5.7 shows the average cost and standard deviation (in brackets) of the best solution found by an EA using either random selection, greedy selection, or tournament selection. All EAs use a random initial population of size $N = 100$ and are stopped after $200n$ search steps. For each combination of problem and solution, we perform 10 independent runs. We only present results for Euclidean test instances as results for random test instances are analogous.

The results indicate a high performance of the proposed crossover operators. Using c_e and c_p with a tournament selection yields better results than the coresponding greedy versions. In contrast, for dynamic costs c_d greedy edge selection results in a better performance. These are also the best results overall. The heuristic bias of the greedy edge selection is stronger and therefore only results in good search

Table 5.7: Performance comparison of crossover operators

		Random	Greedy Selection			Tournament Selection		
n	r		c_e	c_p	c_d	c_e	c_p	c_d
25	0.2	96,438	115,592	115,608	**89,447**	103,214	97,733	91,718
		(2,261)	(2,150)	(2,164)	(1,464)	(2,036)	(1,617)	(1,902)
50	0.2	457,422	528,510	529,245	**432,578**	484,806	468,727	441,824
		(4,918)	(6,185)	(6,183)	(3,644)	(4,898)	(4,832)	(4,482)
75	0.2	1,101,737	1,245,419	1,245,513	**1,053,387**	1,157,656	1,130,829	1,071,152
		(7,630)	(12,385)	(11,534)	(6,255)	(8,070)	(7,769)	(6,684)
100	0.2	2,034,559	2,267,051	2,266,549	**1,958,968**	2,129,021	2,088,810	1,986,557
		(10,219)	(18,686)	(18,146)	(8,129)	(11,030)	(11,388)	(9,456)
25	1.0	20,136	24,055	24,058	**18,721**	21,198	20,442	19,192
		(460)	(435)	(450)	(291)	(407)	(345)	(378)
50	1.0	93,179	107,645	107,623	**88,138**	97,739	95,351	89,989
		(1,044)	(1,213)	(1,268)	(743)	(938)	(962)	(891)
75	1.0	222,889	251,930	251,903	**213,053**	232,829	228,568	216,670
		(1,563)	(2,363)	(2,388)	(1,194)	(1,636)	(1,574)	(1,444)
100	1.0	410,351	457,099	457,119	**394,780**	427,286	420,919	400,443
		(2,110)	(3,703)	(3,830)	(1,708)	(2,245)	(2,204)	(1,890)
25	5.0	23,604	28,498	28,476	**22,313**	23,882	23,672	22,722
		(450)	(480)	(486)	(255)	(329)	(304)	(338)
50	5.0	100,755	117,039	117,011	**95,266**	102,458	101,604	97,147
		(1,015)	(1,367)	(1,318)	(751)	(953)	(915)	(892)
75	5.0	234,973	266,177	266,198	**223,954**	239,479	237,657	227,772
		(1,582)	(2,608)	(2,556)	(1,192)	(1,553)	(1,532)	(1,476)
100	5.0	427,104	476,482	476,491	**409,776**	435,541	432,823	415,940
		(2,163)	(4,025)	(3,918)	(1,720)	(2,172)	(2,209)	(2,028)

Table 5.8: Runtimes of EAs using different crossover operators

	Random	Greedy Selection			Tournament Selection		
n		c_e	c_p	c_d	c_e	c_p	c_d
10	0.04	0.03	0.03	0.04	0.04	0.04	0.04
25	0.25	0.19	0.19	0.25	0.24	0.24	0.27
50	1.08	0.83	0.83	1.00	1.04	1.04	1.09
75	2.81	2.22	2.22	2.51	2.68	2.69	2.75
100	5.53	4.53	4.54	4.95	5.32	5.37	5.39

performance, if the bias matches the characteristics of the problem. All results are significant using a ranked t-test.

Using a heuristic crossover based on edge weights or expected costs does not improve search performance. In contrast, in most instances these configurations are worse than using random edge selection, since the intercosts are not considered. Only relying on distance weights creates solutions similar to the MST, but the edge pairs might generate high intercosts. The expected costs are also not a good approximation of the costs an edge generates (compare the construction heuristics in Sect. 5.3). This is a perfect example of the fact that the bias of an operator has to match the properties of high-quality solutions. It is important to thoroughly investigate the problem and the operators used before designing biased optimization methods. Biasing in the wrong direction or too strongly can be worse than using no bias at all.

Table 5.8 lists the average CPU times of the different EAs in seconds. The experiments were run on a Intel Core i5 Linux machine with 4GB memory. The differences between the configurations are small. Using a greedy edge selection is faster as tournament selection or random edge selection. During crossover, the selected edges are not always eligible for inclusion and new edges need to be selected. In the greedy case, we have a sorted list of all edges and subsequent selections are faster compared to the tournament selection case. This explains the lower runtimes.

5.6 Comparison to the state-of-the-art

We compare the biased approaches from Sects. 5.4 and 5.5 with the state-of-the-art approaches for the QMST problem, namely an EA using the Edge Window Decoder (EWD) representation by Soak, Corne, and Ahn (2006) and an artificial bee colony (ABC) algorithm by Sundar and Singh (2010).

The EA proposed by Soak, Corne, and Ahn (2006) uses the EWD representation, which encodes trees as a list of nodes of length $2(n-1)$. Recombination and mutation operators are directly applied to the list of nodes; A so-called tree construction routine (TCR) constructs a tree from the list of nodes. Soak, Corne, and Ahn (2006) report best performance for real-world tournament selection (RWTS), adjacent node crossover (ANX), and the degree-constrained Kruskal tree construction routine (dK-TCR). The generational EA uses a population size of $N = 100$; crossover and mutation probability is set to $p = 0.6$.

Sundar and Singh (2010) use an artificial bee colony algorithm (ABC). ABC is a population-based approach that starts with a set of initial solutions that are either randomly created or results of some construction heuristic. In each iteration one new neighboring solution for each solution in the population is created. If this new solu-

tion is better than the original, it replaces the original. Then, a second population is created, the so-called onlooker bees. Each onlooker randomly selects one solution out of the first population using a fitness-proportional selection. The onlooker bees create neighboring solutions to the selected solution. Because of the fitness-proportional selection, more promising solutions are explored more thoroughly. The solutions in the first population are replaced by the best solution found by associated onlooker bees. In our implementation of the algorithm, all parameter settings were used as provided in their paper. After finishing an ABC run, Sundar and Singh (2010) used a simple local search to improve the found solutions. We decided to not use this local search at the end, since we were only interested in the performance of the metaheuristic. This ensures a fair comparison.

Table 5.9 shows the mean and standard deviation of the cost of the best solution found and Table 5.10 the average running time (in s). As in the previous experiments, we used a Intel Core i5 Linux machine with 4GB memory. SA denotes the heuristic simulated annealing approach from Sect. 5.4 using dynamic cost c_d. ES denotes the best-performing evolutionary algorithm from Sect. 5.5 with greedy edge selection using dynamic cost. Both biased approaches, SA and EA are significantly better ($p < 0.001$ using a t-tests on the ranks) than the ABC and EWD approach. The type of QMST instances has some influence on the performance. For larger instances ($n > 25$), the best average results are reached using simulated annealing. The comparison of ABC and EWD confirms the results from the literature: ABC gives a better and more robust results compared to EWD, the average costs and the average deviation is significantly lower. Regarding the runtimes, SA and ES are 10% to 25% faster than ABC and EWD for the larger instances ($n \geq 50$). ABC is the slowest of the four approaches.

5.7 Conclusions

The evaluation function of the QMST problem consists of two parts: the sum of the edge costs and the sum of the intercosts, which are the costs that arise for every pair of edges in a solution. Depending on the value-range of the costs and intercosts, the structure of a QMST instance changes. We introduced two constants (w_e and w_i) to the evaluation function to better describe this behavior. If the ratio $r = w_e/w_i$ is high, the edge distances are more important, and solutions are similar to those of the much easier MST problem. In contrast, with a low ratio r, the intercosts become more important. Efficient optimization methods for the QMST problem should consider this ratio.

In a first step, we created different construction heuristics, where each one was biased towards low-weight edges weighted differently. The analysis of the construction

Table 5.9: Comparison with the state-of-the-art

n	r	EWD	ABC	SA c_d	ES c_d
25	0.2	105,573 (2,313)	95,931 (1,455)	91,556 (2,167)	**89,528** (1,496)
50	0.2	493,197 (4,847)	461,229 (3,006)	**430,420** (4,736)	432,505 (3,693)
75	0.2	1,174,388 (7,294)	1,112,207 (4,840)	**1,038,158** (7,282)	1,053,300 (5,858)
100	0.2	2,153,686 (10,111)	2,054,431 (6,476)	**1,923,515** (9,383)	1,958,556 (8,500)
25	1.0	21,693 (439)	20,053 (292)	19,219 (437)	**18,736** (302)
50	1.0	99,602 (1,016)	93,970 (611)	**87,769** (969)	88,140 (760)
75	1.0	236,213 (1,554)	224,996 (930)	**210,057** (1,430)	212,993 (1,196)
100	1.0	432,381 (2,102)	414,227 (1,290)	**387,833** (1,812)	394,896 (1,614)
25	5.0	24,144 (369)	23,420 (304)	22,812 (472)	**22,290** (256)
50	5.0	103,887 (1,005)	101,213 (690)	95,481 (1,040)	**95,239** (735)
75	5.0	242,608 (1,555)	236,584 (1,066)	**221,915** (1,524)	223,792 (1,225)
100	5.0	440,978 (2,183)	430,434 (1,430)	**404,412** (1,983)	409,729 (1,659)

Table 5.10: Runtimes of state-of-the-art comparison

n	EWD	ABC	SA c_d	ES c_d
10	0.02	0.02	0.03	0.04
25	0.18	0.13	0.16	0.26
50	0.89	0.81	0.75	1.00
75	2.73	2.90	2.03	2.45
100	5.96	6.88	4.57	5.00

heuristics performance revealed that the most promising ones are heuristics that are biased towards low-weight edges that have low intercosts with the edges already contained in the solution.

In subsequent steps, we use this knowledge to create problem-specific mutation and crossover operators for the QMST problem. Using these operators, we are able to increase search performance, compared to the use of other operators found in the literature. In the last set of experiments, we compared an evolutionary algorithm and a simulated annealing approach using the new operators to the current state-of-the-art approaches. Experimental results show the superiority of our new approaches. Results are 6 to 12 percent better, depending on the parameters of the problem instance.

References

Assad, A. and W. Xu (1992). "The quadratic minimum spanning tree problem". In: *Naval Research Logistics* 39.3, pp. 399–417.

Bern, M. W. (1988). "Two probabilistic results on rectilinear steiner trees". In: *Algorithmica* 3.2, pp. 191–204.

Bonissone, P., R. Subbu, N. Eklund, and T. Kiehl (2006). "Evolutionary algorithms + domain knowledge = real-world evolutionary computation". In: *IEEE Transactions on Evolutionary Computation* 10.3, pp. 256–280.

Chao, T.-H. and C. Hsu (1994). "Rectlinear steiner tree construction by local and global refinement". In: *IEEE Transactions on Computer-Aided Design of Integrated Circutits and Systems* 13.3, pp. 303–309.

Christofides, N. (1976). *Worst-case analysis of a new heuristic for the traveling salesman problem.* Tech. rep. 388. Pittsburgh, PA: Graduate School of Industratial Administration, Carnegie-Mellon University.

Droste, S. and D. Wiesmann (2003). "Advances in evolutionary computing". In: ed. by A. Ghosh and S. Tsutsui. New York, NY, USA: Springer-Verlag New York, Inc. Chap. On the design of problem-specific evolutionary algorithms, pp. 153–173.

Geisberger, R. and C. Vetter (2011). "Efficient routing in road networks with turn costs". In: *Proceedings of the 10th international conference on Experimental algorithms, Crete, Greece.* SEA'11. Berlin, Heidelberg: Springer-Verlag, pp. 100–111.

Hauschild, M. W., M. Pelikan, K. Sastry, and D. E. Goldberg (2012). "Using previous models to bias structural learning in the hierarchical boa". In: *Evolutionary Computation* 20.1, pp. 135–160.

Ho, J.-M., G. Vijayan, and C. K. Wong (1990). "New algorithms for the rectilinear steiner tree problem". In: *IEEE Transactions on Computer-Aided Design* 9, pp. 185–193.

Hwang, F. K. (1976). "On steiner minimal trees with rectilinear distance". In: *SIAM J. Applied Math.* 1, pp. 104–114.

Julstrom, B. A. and G. R. Raidl (2002). "Initialization is robust in evolutionary algorithms that encode spanning trees as sets of edges". In: *SAC '02: Proceedings of the 2002 ACM symposium on applied computing, Madrid, Spain.* New York, NY, USA: ACM, pp. 547–552.

Karaboga, D. and B. Basturk (2007). "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm". In: *Journal of Global Optimization* 39 (3), pp. 459–471.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). "Optimization by simulated annealing". In: *Science* 220, pp. 671–680.

Öncan, T. and A. P. Punnen (2010). "The quadratic minimum spanning tree problem: a lower bounding procedure and an efficient search algorithm". In: *Computers and Operations Research* 37.10, pp. 1762–1773.

Palubeckis, G., D. Rubliauskas, and A. Targamadze (2010). "Metaheuristic Approaches for the Quadratic Minimum Spanning Tree Problem". In: *Information Technology and Control* 39.4, pp. 257–268.

Raidl, G. R. and B. A. Julstrom (2003a). "Edge sets: an effective evolutionary coding of spanning trees". In: *IEEE Transactions on Evolutionary Computation* 7.3, pp. 225–239.

Richards, D. (1989). "Fast heuristic algorithms for rectilinear steiner trees". In: *Algorithmica* 4, pp. 191–207.

Robins, G. and A. Zelikovsky (2005). "Tighter bounds for graph steiner tree approximation". In: *SIAM J. Discret. Math.* 19.1, pp. 122–134.

Rothlauf, F. (2009c). "On the Bias and Performance of the Edge-Set Encoding". In: *IEEE Transactions on Evolutionary Computation* 13.3, pp. 486–499.

Rothlauf, F. (2011). *Design of Modern Heuristics: Principles and Application.* 1st. Springer.

Soak, S.-M., D. W. Corne, and B.-H. Ahn (2006). "The edge-window-decoder representation for tree-based problems". In: *IEEE Transactions on Evolutionary Computation* 10.2, pp. 124–144.

Steitz, W. and F. Rothlauf (2012). "Edge orientation and the design of problem-specific crossover operators for the ocst problem." In: *IEEE Transactions on Evolutionary Computation* 16.1, pp. 108–116.

Sundar, S. and A. Singh (2010). "A swarm intelligence approach to the quadratic minimum spanning tree problem". In: *Information Sciences* 180.17, pp. 3182–3191.

Winter, S. (2002). "Modeling costs of turns in route planning". In: *GeoInformatica* 6 (4), pp. 345–361.

Wolpert, D. H. and W. G. Macready (1997). "No free lunch theorems for optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1, pp. 67–82.

Zhou, G. and M. Gen (1998). "An effective genetic algorithm approach to the quadratic minimum spanning tree problem". In: *Computers and Operations Research* 25 (3), pp. 229–237.

Chapter 6

New heuristic approaches for the bounded-diameter spanning tree Problem

Wolfgang Steitz

Abstract

Given a bound, the bounded-diameter spanning tree (BDMST) problem seeks a spanning tree of minimum total weight with a diameter not exceeding the given diameter bound. Depending on the tightness of the bound, optimal solutions have different structures. For loose bounds, optimal solutions are similar to the much easier minimum spanning tree problem, and greedy heuristics perform best. In contrast, these approaches fail for tight diameter bounds. This paper investigates how the structure of good solutions, and in particular their backbones, change depending on the diameter bound. Two new heuristics are then designed to overcome the shortcomings of existing approaches; required parameters are investigated; and the paper presents performance results for various BDMST instances.

6.1 Introduction

The bounded-diameter minimum spanning tree (BDMST) problem is an $\mathcal{NP}$-hard combinatorial optimization problem with many real-world applications, such as communication network design or transportation problems. The problem is an extension of the much simpler minimum spanning tree problem. Given an undirected, weighted and connected graph, the goal is to identify a spanning tree with minimum total costs and a diameter not exceeding the given diameter bound of the instance. A wide variety of approaches to solving the BDMST can be found in the literature. Since exact approaches (Gruber and Raidl, 2010; A. d. Santos, Lucena, and Ribeiro, 2004) only work for small problem instances, heuristics have been proposed. Besides metaheuristics such as evolutionary algorithms (Raidl and Julstrom, 2003b; Singh and Gupta, 2007) or ant colony optimization (Gruber, Hemert, and Raidl, 2006), some problem-specific construction heuristics were proposed (Abdalla and Deo, 2002; Deo and Abdalla, 2000; Gruber and Raidl, 2009; Julstrom, 2009; Raidl and Julstrom, 2003b; Requejo and E. Santos, 2009).

Existing construction heuristics are often based on Prim's minimum spanning tree (MST) algorithm (Prim, 1957). They start from a root node and iteratively connect the remaining nodes with preferably low-weight edges, without violating the diameter bound. The very greedy ones – the one-time tree construction (OTTC) and the center-based tree construction (CBTC) – only work well for instances with loose diameter bounds. For tight diameter bounds, these algorithms show low performance. OTTC and CBTC search too greedily as they always select the shortest edges, and thus, are unable to create backbones (the tree without its leaf nodes) that span large areas (Julstrom, 2009). The randomized center-based tree construction (RTC) algorithm overcomes this problem by randomly choosing nodes to connect next, and then connecting them using the shortest possible edge. While this works better for tight diameter bounds, RTC's performance drops for larger diameter bounds. Overall, there is no known construction heuristic that works well for all types of BDMST instances.

The goal of this paper is to design new construction heuristics for the BDMST problem, based on an experimental analysis of the existing problems and approaches. We analyze the backbone of optimal solutions and compare these with backbones created by the existing approaches. The results give us a better understanding of these heuristics' shortcomings. So that, we are able to design two new heuristic approaches, which yield better backbones and show a better overall performance. The main findings are:

1. An experimental study of the backbones created by the existing approaches (CBTC, OTTC and RTC) explains why their performance varies with the type of BDMST instance (tight diameter bound versus loose diameter bound).

2. We propose the construction heuristics *node selection tree construction* (NSTC) and *savings tree construction* (STC), that are able to create backbones with similar properties to those of optimal solutions.

3. Experiments on a set of larger Euclidean BDMST instances show that our operators NSTC and STC are able to create trees with significantly lower costs than existing approaches.

Section 6.2 describes the BDMST problem and gives an overview on existing optimization approaches. The following Section 6.3 reviews OTTC, CBTC and RTC and studies the properties of trees created by those methods. In Section 6.4, two new approaches NSTC and STC are introduced. We show that these algorithms produce trees with backbone structures similar to those of optimal solutions. The experiments in Section 6.5 compare the performance of all construction heuristics for large instances of Euclidean BDMST problems with up to $n = 1000$ nodes. The final section summarizes the paper and gives an outlook on future work.

6.2 The BDMST problem

In a graph, the eccentricity of a node v is the maximum number of edges on the shortest path from v to every other node. The maximum eccentricity of all nodes in a graph is called the diameter. For trees, there is a unique path between every pair of nodes. Therefore, the eccentricity of a node v is just the number of edges on the path from v to every other node in the tree. The diameter of a tree is the number of edges on any path between two nodes. The center of a tree is the unique node (if the tree's diameter is even) or the pair of adjacent nodes (if the diameter bound is odd) with minimum eccentricity.

Formally, let $G = (V, E)$ be a weighted, undirected graph with $n = |V|$ nodes, $m = |E|$ edges and associated real-valued distance weights $d_{uv} \geq 0$ for each pair of nodes. Given an integer bound $D \in \{2, \ldots, n-1\}$, a bounded-diameter spanning tree is a spanning tree $T = (V, E_T)$ on G with diameter no more than D and $E_T \subseteq E$. The total cost $c(T)$ of a spanning tree T is the sum of the real-valued weights of the tree's edges:

$$c(T) = \sum_{(u,v) \in E_T} d_{uv}$$

The bounded-diameter spanning tree problem (BDMST) seeks a bounded-diameter spanning tree on G with minimum total costs for a given diameter bound D.

Like many other constrained graph problems this problem is $\mathcal{NP}$-hard for $4 \leq D < n-1$ (Garey and Johnson, 1979). Only small problem instances can be solved

by exact approaches in a reasonable amount of time. Achuthan et al. (1994) developed branch-and-bound algorithms for the BDMST problem and were able to solve instances with up to 100 nodes, but only for incomplete graphs. A. d. Santos, Lucena, and Ribeiro (2004) improved this approach and were able to solve problems up to 100 nodes and 1000 edges.

Gouveia and Magnanti (2003) described a network flow model. Gruber and Raidl (2005) described a 0-1 integer linear programming approach for the problem. Gruber and Raidl (2010) proposed a new ILP formulation utilizing jump inequalities to ensure the diameter constraint and solve it with branch-and-cut. As for approximation methods, Kortsarz and Peleg (1999) showed that, unless $\mathcal{P} = \mathcal{NP}$, there is no $o(logn)$-approximation.

For larger instances, researchers applied metaheuristics like evolutionary algorithms (Raidl and Julstrom, 2003b; Singh and Gupta, 2007), variable neighborhood search (Gruber, Hemert, and Raidl, 2006) and ant colony optimization (Gruber, Hemert, and Raidl, 2006) to the BDMST problem. Lucena, Ribeiro, and A. C. Santos (2009) proposed a hybrid GRASP and ILS approach. Raidl and Gruber (2008) combined variable neighborhood search with Lagrangian relaxation and showed promising results for complete graphs with up to 81 nodes. Besides those metaheuristics, some fast and simple construction heuristics were proposed (Abdalla and Deo, 2002; Deo and Abdalla, 2000; Gruber and Raidl, 2009; Julstrom, 2009; Raidl and Julstrom, 2003b; Requejo and E. Santos, 2009). The three best-performing construction heuristics are presented in greater detail in the following section.

6.3 Existing construction heuristics for the BDMST problem

6.3.1 One-time tree construction – OTTC

Abdalla and Deo (2002) proposed a construction heuristic called *one-time tree construction* (OTTC) based on Prim's algorithm. It starts at an arbitrary node and generates a tree by iteratively appending edges with the lowest weight, which connects the tree with a node that is not in the tree without violating the diameter bound.

As in the usual implementation of Prim's algorithm, the one-dimensional help arrays *near* and *wnear* are used. For each node u not connected to the partial tree, $near[u]$ is the eligible tree node nearest to u and $wnear[u]$ is the weight of the edge connecting u and $near[u]$. A node is called eligible if appending an edge to it does not violate the diameter bound. Another array $ecc[u]$ holds the current eccentricity of node u, if u is part of the tree, $ecc[u] = -1$ otherwise. The array

$ecc[\cdot]$ is updated when a new node joins the partial tree. OTTC ensures the diameter bound by limiting the eccentricity of each node to D, since the diameter of a tree is the maximum eccentricity of its nodes. A matrix $dist[u][v]$ holds the number of edges on the path from u to v for each pair of nodes both in the spanning tree, $dist[u][v] = -1$ otherwise. The array $dist[\cdot]$ helps with updating the eccentricities, as $ecc[u] = \max(dist[u][\cdot])$.

After inserting a new edge, all four arrays are updated, overall $n-1$ updates are performed. Since updating *near* and *wnear* has a worst case complexity of $O(n^2)$, OTTC has a time complexity of $O(n^3)$. The solution quality of the trees created by OTTC strongly depends on the choice of the start node. Therefore, the algorithm is run from each node to obtain a low-weight bounded-diameter spanning tree. This increases the worst-case time to $O(n^4)$.

6.3.2 Center-based tree construction – CBTC

Julstrom (2009) proposed a Prim-based construction heuristic called *center-based tree construction* (CBTC). CBTC starts the tree construction at a center node and iteratively appends edges. This algorithm uses the fact that in a tree with diameter D, no node is more than $\lfloor D/2 \rfloor$ edges away from the center (Handler, 1973). The number of edges on the path from the center to each node is called the node's depth. To meet the diameter bound, the algorithm bounds the depth of each node. As before, CBTC uses the help arrays $near[\cdot]$ and $wnear[\cdot]$. Additionally, the array $depth[u]$ holds the depth of each tree node u.

Algorithm 3 outlines the functionality of CBTC. With an even diameter bound, a single node v_0 is used as the center of the tree. For odd D, the center consists of two nodes (v_0 and v_1) that are connected by an edge. One of those center nodes is the start node, the other is the nearest node. The heuristic iteratively extends the tree with the edge of smallest weight that connects a new node v to a tree node $near[v]$ whose depth is less than $\lfloor D/2 \rfloor$. The depth of the new node v is $depth[near[v]] + 1$.

The running time of CBTC is $O(n^2)$. Like OTTC, the algorithm strongly depends on the start node v_0 and is run from each node in the tree, resulting in an overall running time of $O(n^3)$ (Julstrom, 2009).

6.3.3 Randomized center-based tree construction – RTC

Both OTTC and CBTC are very greedy, as they always connect the node nearest to the current solution. For Euclidean instances with tight diameter bounds, this behavior results in a low performance. To overcome this, Julstrom (2009) proposed a randomized version of CBTC called *randomized center-based tree construction*

Algorithm 3 CBTC

```
 1: if D is even then
 2:     depth[v_0] ← 0
 3:     T ← ∅
 4: else
 5:     v_1 ← node nearest to v_0
 6:     depth[v_0] ← 0
 7:     depth[v_1] ← 0
 8:     T ← {(v_0, v_1)}
 9: initialize the arrays near[·] and wnear[·]
10: while |T| < n − 1 do
11:     v ← node not in T with smallest wnear[v]
12:     T ← T ∪ {(v, wnear[v])}
13:     depth[v] ← 1 + depth[near[v]]
14:     if depth[v] < ⌊D/2⌋ then
15:         for all node u ∉ T do
16:             if d_{u,v} < wnear[u] then
17:                 near[u] ← v
18:                 wnear[u] ← d_{u,v}
    return T
```

(RTC). New nodes are not selected in a greedy manner, but at random. This approach was first described by Raidl and Julstrom (2003b).

The functionality of RTC is analog to that of CBTC shown in Algorithm 3; only lines 5 and 11 are changed. Instead of choosing the nearest node (smallest *wnear*), the next node is selected randomly. Tree construction starts from a randomly selected starting node. In the odd diameter case, the second center node is also selected at random. Subsequent nodes are also selected at random, but still connected using the lowest-weight edge. The RTC algorithm is much simpler than CBTC, but the running time remains unchanged at $O(n^2)$. Julstrom (2009) suggests running RTC n times, which results in a time complexity of $O(n^3)$.

6.3.4 Analysis and comparison

Julstrom (2009) compared and analyzed the performance of OTTC, CBTC and RTC. For Euclidean instances with large diameter bounds OTTC and CBTC performed best, since the solutions are similar to the MST. Therefore, the greediness of OTTC and CBTC leads to low-weight trees. In contrast, for instances with tighter diameter bounds, RTC outperformes OTTC and CBTC. Julstrom (2009) explains this behavior by the fact that the OTTC and CBTC algorithms behave too greedily. Let the backbone of a tree be the tree without its leaf nodes and without the

edges connecting them. The backbones created by OTTC and CBTC consist of very short edges. If the diameter is tight, many nodes have to be connected to the backbone via longer edges. RTC chooses the nodes in a non-greedy manner, which results in backbones that span larger parts of the graph. This allows leaf nodes to be connected with shorter edges.

Creating a low-weight backbone that also spans a large area is the crucial part for the heuristic construction of good solutions for the BDMST problem. In the following experiments we compare the backbones created by the tree construction heuristics with those of optimal solutions for Euclidean BDMST instances. To describe a backbone, we analyze its size (i.e., the number of nodes contained in the backbone), its total weight as well as the area covered by the backbone. To quantify the covered area, we determine the convex hull of the nodes contained in the backbone and calculate the area of the resulting polygon.

For small instances ($n = 40$), we also show results for optimal solutions, which are identified using the integer linear program (ILP) approach proposed by Gruber and Raidl (2005). Using Gurobi 4.6, we were able to determine optimal solutions for Euclidean BDMST instances for $n = 40$ in reasonable time. Following Julstrom (2009) and Gruber, Hemert, and Raidl (2006), we use the graphs of the Euclidean Steiner Problem found in the Beasley's OR-library (Beasley, 1990)[1]. For each graph and each diameter bound D, all three heuristics were run once.

The plots on the left in Figure 6.1 show the results for $n = 40$. For instances with $n < 40$ the results are equivalent to those of $n = 40$. In the plots on the right, we study large instances with $n = 1000$, where we cannot identify optimal solutions in reasonable time. Additionally, the plots show the properties of the MST.

Comparing the backbones created by the heuristics with those of optimal solutions (upper plots), all three construction heuristics create backbones with too few nodes. OTTC and CBTC are very similar, while RTC's backbones are closer to the optimum. Looking at the covered area, we can see that the backbone of optimal solutions covers a large part of the graph. The random selection of nodes from RTC leads to similar results. CBTC and OTTC only cover very small parts of the graph, especially for tight diameter bounds. The weight of the created backbones is low for the very greedy heuristics CBTC and OTTC. This is partly explained by the lower node count of the backbones, but even so, the difference between these weights and those of the optimal solution is significant. The weight of RTC's backbone is similar to that of the optimal solution. Another interesting observation is that the weight of the optimal backbone does not increase with the diameter bound.

For large instances (Figure 6.1, right), we cannot identify optimal solutions. The results of the heuristics are similar to those for small instances. OTTC's behavior is

[1] `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`

similar to that of CBTC. All measures increase with the diameter bound. For tight diameter bounds they cover only a very small part of the graph, which explains their poor performance for these instances. In contrast, RTC's backbone covers almost the complete graph, independently of the diameter bound. This leads to heavier backbones.

6.4 Two new heuristic approaches

As analyzed in the last section, none of the existing construction heuristics are able to create good backbones and therefore high-quality solutions for all diameter bounds. This section proposes two new construction heuristics. The first one is an extension of the CBTC approach. The second heuristic resembles the savings heuristic used for vehicle routing problems. After introducing the algorithms, we analyze the solutions, especially the backbones they create.

6.4.1 Extending the existing approaches

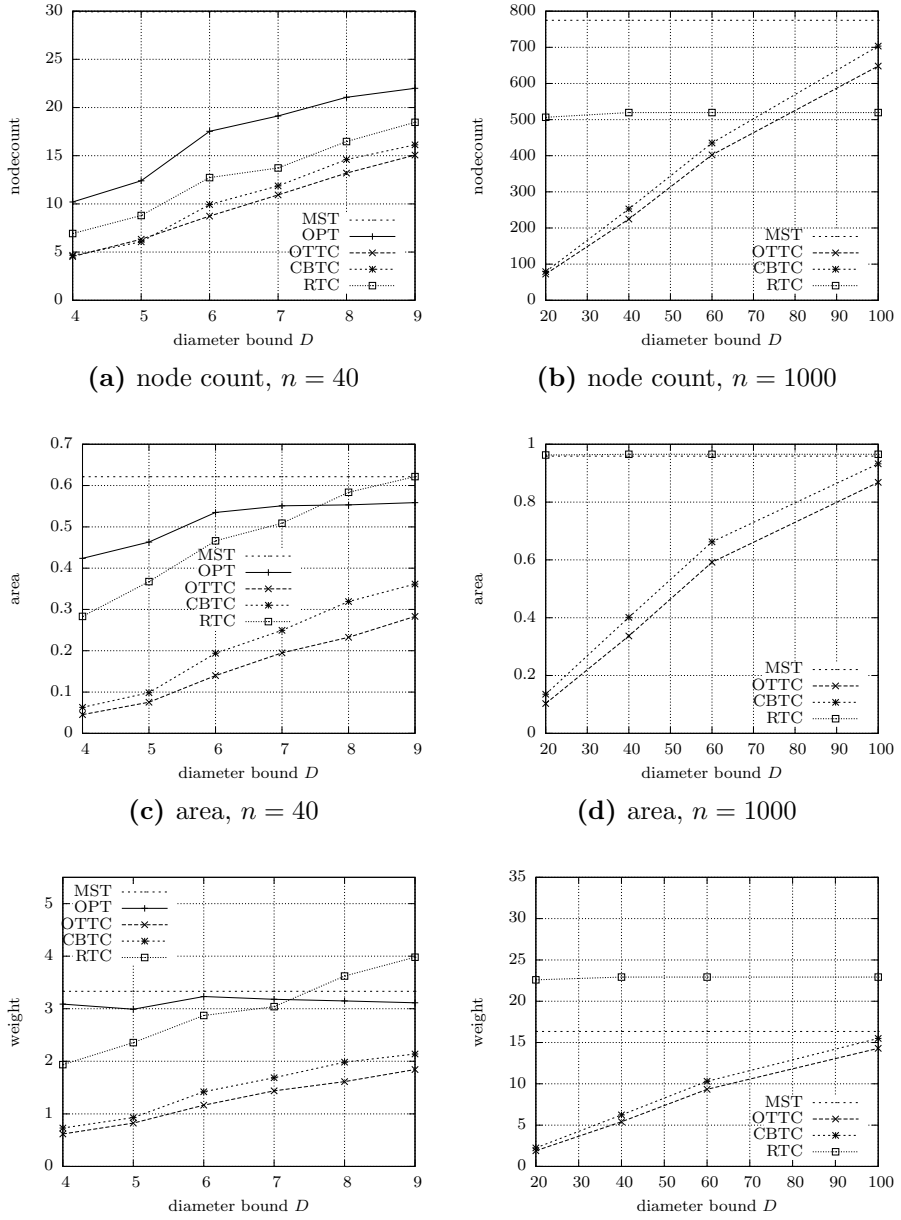
As seen in the previous section, CBTC and RTC differ in how they choose the next node to be connected to the partial tree. CBTC always selects the closest node in a greedy fashion. This works well for loose diameter bounds, but CBTC fails for tight diameter bounds. In contrast, RTC randomly selects the next node, which only works well for large diameter bounds.

Our new approach changes the way the next node is selected. During the node selection, we consider the distance to the partial solution *and* the possible cost savings generated by adding this node. Therefore, when selecting the next node in the algorithm (Algorithm 3, line 11), each unconnected node is weighted using:

$$w(u) = \begin{cases} \lambda \cdot wnear[u] & \text{if } depth(u) = \lfloor D/2 \rfloor \\ \lambda \cdot wnear[u] - \sum\limits_{v \in V, v \notin V_T, wnear[v] > d_{u,v}} (wnear[v] - d_{u,v}) & \text{otherwise,} \end{cases}$$

where parameter λ controls the influence of the distance weight. The first part is just the distance of u to the partial solution. The second part sums up the changes of the *wnear* values that would arise if node u were added to the tree. If a node u already has a depth of $depth(u) = D/2$, no other node can be connected to u and the reduction of the *wnear* values equals zero.

This allows the algorithm to choose nodes that have a larger distance to the current partial solution, if this decreases $wnear[v]$ for many nodes. The purpose of the selection scheme is to build better backbones, that cover larger parts of the



(a) node count, $n = 40$

(b) node count, $n = 1000$

(c) area, $n = 40$

(d) area, $n = 1000$

(e) weight, $n = 40$

(f) weight, $n = 1000$

Figure 6.1: The plots compare the properties of the backbones created by OTTC, CBTC and RTC. For small instances (left), the properties of optimal solutions are also shown. CBTC and OTTC create similar lightweight backbones with few edges. These backbones cover only small parts of the graph. RTC's backbones are more similar to those of optimal solutions.

graph, while still preferring low-weight edges. We call this algorithm *node-selection tree construction* (NSTC).

Unfortunately, this approach increases the complexity of the algorithm from $O(n^2)$ to $O(n^3)$. While this is the same complexity as OTTC, preliminary experiments have shown that the actual runtime of this approach is much higher than that of the other algorithms. To save time, we propose not to run the algorithm from every root node, but to select a good root node in advance and to run NSTC only once. A possible way to select a good root node is to create star trees from every node and choose the root as the center of the star with minimal costs. Comparing all star trees can be done in $O(n^2)$. A fast way is to sum up every row in the distance matrix and determine the row with the lowest sum. This decreases the average performance of NSTC only slightly, compared to running NSTC from every node.

The parameter λ strongly influences the performance of NSTC. For large values of λ, NSTC works more like CBTC, since *wnear* has the most influence on the node selection and is therefore suited for large diameter bounds. For tighter diameter bounds lower values of λ lead to better soutions. To automatically assess the tightness of the instance and set λ , we recommend to calculate the parameter λ as:

$$\lambda = \alpha \cdot \frac{D}{diam_{MST}},$$

where D is the current diameter bound and $diam_{MST}$ is the diameter of the MST. The performance of NSTC is less sensitive to the parameter α as to parameter λ.

The following experiment analyzes the influence of α. We study the performance of NSTC for α values of [0, 2, 4, 6, 7, 8, 9, 10] for Euclidean BDMST instances with $n = 500$ nodes, taken from the OR-library. Figure 6.2 shows the average solution quality over the diameter bound D.

As expected, for small diameter bounds, lower values of α result in the best performance. For larger diameter bounds larger values of α are needed. Setting $\alpha = 7$ results in a good performance for all diameter bounds, in particular for small bounds. In the remainder of the paper, we will always use this setting of α.

6.4.2 The savings heuristic

The second new construction heuristic for the BDMST problem resembles parts of the savings heuristics used in vehicle routing problems. Starting from a star-tree, we check each node to see if there is a cheaper connection that does not violate the bound. Like CBTC, RTC and NSTC, we ensure the diameter bound by building a tree from a root node and limiting the height of each node to $\lfloor D/2 \rfloor$. We denote this algorithm as *savings tree construction* (STC).

Algorithm 4 describes the algorithm in detail. At first, we connect every node with the root v_0 forming a star-tree. In the odd diameter case, the node nearest to

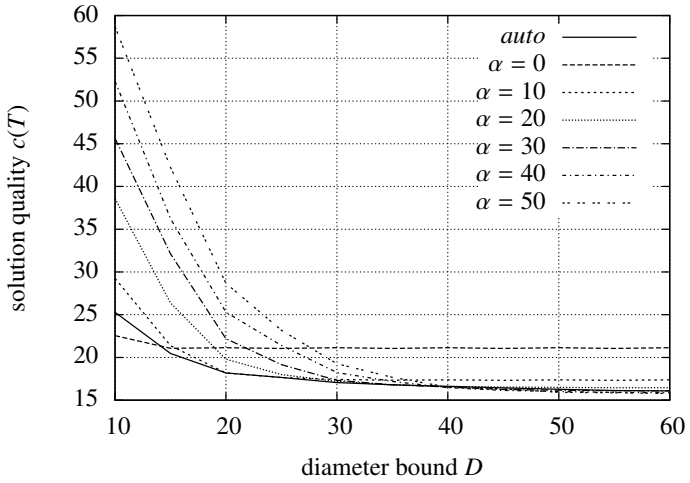


Figure 6.2: Comparison of different settings of α for Euclidean instances of size $n = 500$. Setting $\alpha = 7$ yields the best results independently of the diameter bound.

the root node becomes the second root and every node is connected to one of the root nodes, depending on the distance. The array $roots[\cdot]$ stores the root of each tree node. After this initialization, the algorithm checks each node v to see if there is a better connection. The edge $(v, roots[v])$ is removed from T. This splits the tree into two subtrees. We calculate the height h of the subtree rooted in v. Then, we determine the eligible node u with the shortest distance $d(u, v)$. A node is eligible if $depth[u] + h < D/2$. Before reconnecting the tree, the algorithm updates the depth of all nodes connected to v, then the edge (u, v) is added to T.

The algorithm's complexity is $O(n^2)$. The initialization step is done in linear time. The improvement phase has quadratic complexity. As the existing approaches presented in Section 6.3, the choice of the root node has a strong influence on the solution quality. We recommend running STC from every tree node once, increasing the complexity to $O(n^3)$.

Preliminary experiments showed, that in the improvement phase the order of the nodes has a strong influence on the solution quality. In the following experiment, we compare three different ordering schemes: a random ordering scheme (denoted *rnd*), an ascending (*asc*) ordering scheme and a descending (*desc*) ordering scheme – the latter two determined according to the distance to the root node. Table 6.1 lists the average solution quality and the corresponding standard deviation for Euclidean BDMST instances. As in the previous experiments, we use the instances from the OR-library.

In most cases, the descending ordering produces the best solutions. Ascending ordering is significantly worse for all problem sizes and diameter bounds. However,

Algorithm 4 STC

```
1:  // Initialization phase
2:  T ← {}
3:  if D is even then
4:      depth[v_0] ← 0
5:      for all node u not in T do
6:          T ← T ∪ {(v_0, u)}
7:          root[u] ← v_0
8:  else // Odd diameter
9:      v_1 ← node nearest to v_0
10:     depth[v_0] ← 0
11:     depth[v_1] ← 0
12:     T ← {(v_0, v_1)}
13:     for all node u not in T do
14:         if d_{v_0,u} < d_{v_1,u} then
15:             T ← T ∪ {(v_0, u)}
16:             root[u] ← v_0
17:         else
18:             T ← T ∪ {(v_1, u)}
19:             root[u] ← v_1
20:  // Improvement phase
21: nodes ← list of nodes without root nodes
22: for all node v in nodes do
23:     T ← T \ {(v, roots[v])}  // remove edge
24:     h ← height(T, v)
25:     u ← node u in T with lowest d_{u,v} and depth[u] + h < D/2
26:     depth[v] ← depth[u] + 1
27:     update depth of all nodes connected to v
28:     T ← T ∪ {(u, v)}  // add edge
    return T
```

Table 6.1: Performance of the STC heuristic using different sorting schemes

n	D	rnd		asc		desc	
100	5	17.798	(1.143)	17.389	(1.036)	**17.289**	(0.899)
	10	10.211	(0.487)	10.633	(0.473)	**9.689**	(0.376)
	15	9.089	(0.354)	9.535	(0.314)	**8.258**	(0.321)
	25	8.371	(0.322)	8.765	(0.240)	**7.272**	(0.245)
250	10	**19.329**	(0.671)	21.209	(0.478)	19.394	(0.496)
	15	16.188	(0.679)	17.709	(0.588)	**15.682**	(0.393)
	20	14.626	(0.422)	15.628	(0.398)	**13.564**	(0.301)
	40	13.234	(0.421)	13.369	(0.226)	**11.197**	(0.187)
500	15	**25.369**	(0.760)	29.298	(0.825)	26.670	(0.685)
	30	19.684	(0.507)	21.272	(0.461)	**18.373**	(0.332)
	45	18.406	(0.408)	19.149	(0.320)	**16.423**	(0.241)
	60	18.133	(0.580)	18.139	(0.257)	**15.661**	(0.277)
1000	20	**35.089**	(0.840)	41.895	(1.091)	37.241	(0.675)
	40	27.724	(0.530)	30.589	(0.425)	**26.381**	(0.300)
	60	26.000	(0.527)	26.947	(0.366)	**23.538**	(0.224)
	100	24.947	(0.431)	24.711	(0.200)	**21.977**	(0.187)

the random ordering seems to work better for tight diameter bounds, in particular for the larger instances. But overall, ordering the nodes in descending order according to their distance to the root node seems to be the best choice. This means that the algorithm works from the outer regions of the graph to the root node. We recommend using this sorting scheme; the following experiments will all use the descending ordering.

6.4.3 Analysis and comparison

In a similar manner to our analysis of existing approaches (Section 6.3.4), we use the following experiments to analyze the backbones created by NSTC and STC. We use the same set of BDMST instances as in the previous experiments and also compare with respect to the size, the covered area, and the total weight of the backbone.

The plots in Figure 6.3 show the experimental results. For the small instances (plots on the left), the results of NSTC are very similar to those of CBTC and OTTC (see Figure 6.1). All measurements increase with the diameter bound, but the lightweight backbone is not able to span large parts of the graph. Backbones created by STC are obviously different. The node-count is slightly higher than in optimal solutions, therefore the weight of the backbone is also higher. The covered area is very similar to that of optimal backbone. For large instances (Figure 6.3,

right), both STC and NSTC create similar backbones that are different from those of CBTC/OTTC, but similar to those of RTC. The node counts are very high, which results in a large covered area. Also, the total weights of the backbones are lower than those of RTC, and more nodes are part of the backbone. Since less of the remaining nodes need to be connected, this should result in lower total costs.

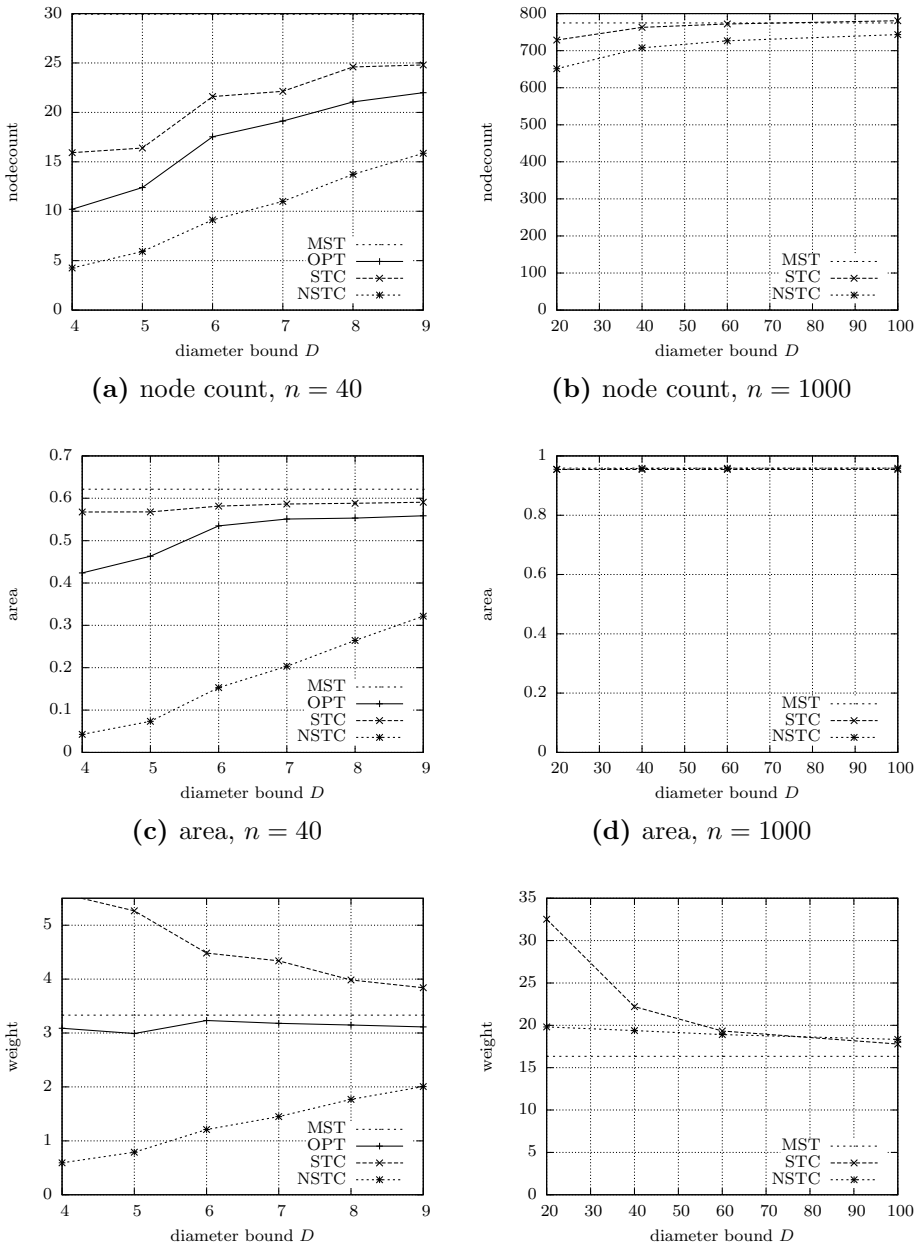
6.5 Performance analysis for large scale Euclidean instances

In this section we compare the performance of the three existing construction heuristics presented in Section 6.3 and the two new heuristics proposed in Section 6.4 on large BDMST instances up to $n = 1000$ nodes. Following Julstrom (2009) and Gruber, Hemert, and Raidl (2006), we use the graphs of the Euclidean Steiner Problem found in the Beasley's OR-library (Beasley, 1990). Additionally, we randomly create Euclidean graphs, by placing the nodes in the unit square and calculating the Euclidean distances. This approach is analog to the creation of the OR-library instances and was also done by Julstrom (2009) and Gruber, Hemert, and Raidl (2006) for the BDMST problem. For each graph, the diameter bound D is set to four different values. To have 200 instances for each problem size overall, we created 185 instances per size additional to the ones from Beasley.

Table 6.2 shows the average costs of the created solutions and the corresponding standard deviation in brackets. The best performing heuristic for each type of instance is marked in bold. Additionally, the last two columns list the improvement (in percentage) of STC and NSTC over the three existing approaches. The lowest average costs are always reached by using one of the new approaches. Comparing the existing approaches confirms the results from the literature. OTTC and CBTC work best for large bounds, whereas for smaller bounds RTC outperforms.

Looking at the new approaches, NSTC on average outperforms the existing approaches for all instances (significant using a Wilcoxon rank-sum test with an error level of 0.01). The biggest improvements are reached for the large instances with small or medium diameter bounds. The improvements here range from 13% to 27.5%. STC is not able to outperform the existing approaches for tight diameter bounds. For large and medium bounds, STC is able to significantly outperform the other approaches (also significant using a Wilcoxon rank-sum test test with an error level of 0.01). Overall, NSTC is the best choice, if no a-priori knowledge about the type of instance is known. For solving instances with loose diameter bounds, STC is the better method.

Table 6.3 lists the average CPU time of the construction heuristics needed to generate one solution tree. The algorithms have been implemented in C++, compiled



(a) node count, $n = 40$ **(b)** node count, $n = 1000$

(c) area, $n = 40$ **(d)** area, $n = 1000$

(e) weight, $n = 40$ **(f)** weight, $n = 1000$

Figure 6.3: The plots analyze the properties of the backbones created by STC and NSTC. For small instances, NSTC creates solutions similar to CBTC and OTTC, and STC is able to create solutions that are more similar to an optimal backbone. For large instances, both new heuristics create good backbones.

Table 6.2: Performance of construction heuristics for large Euclidean BDMST instances

n	D	mst	OTTC		CBTC		RTC		STC		NSTC		STC %	NSTC %
100	5	6.76	29.01	(2.03)	26.62	(1.43)	20.77	(3.89)	**17.03**	(0.73)	20.02	(1.50)	18.0	3.6
	10	6.76	18.57	(2.13)	15.74	(1.20)	10.66	(0.79)	**9.58**	(0.34)	10.04	(1.35)	10.1	5.7
	15	6.76	12.94	(1.82)	11.16	(0.92)	9.90	(0.46)	**8.21**	(0.27)	9.46	(1.36)	17.1	4.5
	25	6.76	8.19	(0.86)	7.64	(0.47)	9.84	(0.46)	**7.18**	(0.23)	7.69	(0.75)	6.1	-0.6
250	10	10.56	56.70	(4.15)	49.88	(3.27)	19.13	(2.17)	19.45	(0.55)	**15.48**	(0.72)	-1.7	19.1
	15	10.56	43.51	(4.31)	37.07	(2.98)	16.21	(0.52)	15.76	(0.42)	**13.61**	(0.79)	2.8	16.0
	20	10.56	33.14	(4.03)	26.70	(2.40)	15.80	(0.43)	13.58	(0.29)	**12.79**	(0.93)	14.1	19.0
	40	10.56	14.20	(2.12)	12.53	(0.68)	15.80	(0.42)	**11.27**	(0.21)	11.81	(0.88)	10.0	5.8
500	15	14.81	105.43	(6.88)	94.83	(5.04)	23.73	(0.85)	26.87	(0.59)	**19.66**	(0.32)	-13.2	17.1
	30	14.81	57.35	(6.65)	45.80	(3.58)	22.49	(0.47)	18.53	(0.37)	**17.56**	(0.84)	17.6	21.9
	45	14.81	32.45	(5.74)	25.45	(2.47)	22.45	(0.44)	**16.54**	(0.27)	16.66	(1.18)	26.3	25.8
	60	14.81	21.03	(3.79)	17.76	(0.96)	22.50	(0.46)	**15.75**	(0.23)	16.03	(1.15)	11.4	9.8
1000	20	20.83	216.42	(9.44)	196.89	(8.64)	32.34	(0.72)	37.33	(0.69)	**28.14**	(0.34)	-15.4	13.0
	40	20.83	120.69	(16.71)	97.61	(7.67)	31.93	(0.49)	26.37	(0.34)	**25.53**	(0.79)	17.4	20.0
	60	20.83	68.72	(13.11)	50.44	(5.31)	31.93	(0.49)	23.48	(0.25)	**23.16**	(0.62)	26.5	27.5
	100	20.83	27.98	(4.83)	23.39	(0.90)	31.92	(0.49)	21.92	(0.18)	**21.63**	(0.34)	6.3	7.5

Table 6.3: CPU times for large Euclidean BDMST instances

n	OTTC	CBTC	RTC	STC	NSTC
100	0.032	0.010	0.011	0.031	0.003
250	0.491	0.144	0.147	0.413	0.039
500	4.641	1.190	1.179	3.535	0.324
1000	40.931	9.586	10.488	37.537	3.923

with GCC 4.6 and executed on a single core of a 2.67GHz Quad-Core Intel Core i5 Linux machine with 4GB RAM. The comparison of average CPU times shows that OTTC and STC are the slowest heuristics, using about 40 seconds for a 1000 node BDMST instance. CBTC's and RTC's process time is very similar, since the algorithms themselves are very similar. NSTC is the fastest algorithm, since we do not run the algorithm from every possible root node. Running NSTC from every root node would make it significantly slower than the other algorithms. The same approach could be applied to the other methods in order to reduce their CPU time.

6.6 Summary and conclusions

This work investigates construction heuristics for Euclidean bounded-diameter spanning tree problems. The three existing approaches are all based on Prim's minimum spanning tree algorithm and iteratively generate solutions by greedily selecting short edges that do not violate the diameter bound. The performance of these algorithms differs with the type of BDMST instance. For instances with a loose diameter bound, OTTC and CBTC are the best choices, whereas for tight diameter bounds the performance of these two drops sharply. In contrast, RTC works well for small diameter bounds, but is unable to create good solutions if the bound is high.

According to Julstrom (2009), this behavior can be explained by the type of backbones the algorithms create. The backbone of a tree is the tree without its leaf nodes. We conducted an experimental analysis of the backbones created by the three construction heuristics. The results of OTTC and CBTC were very similar. While both create low-weight backbones, these only cover small parts of the graph. The backbones of RTC cover large parts of the graph, but the weight of the backbones is high. The results confirm the explanations offered by Julstrom (2009).

In this paper, we propose two new construction heuristics. NSTC builds on the CBTC and RTC approach. It changes the way the following node in the iterative approach is selected. Instead of choosing the next node at random (RTC) or using a greedy approach (CBTC), NSTC selects the next node according to distance and possible cost savings. The second approach, STC, does not start from an empty

solution, but from a star tree. In a similar way to the savings heuristic used in vehicle routing, we compute the savings of each node generated by not connecting the node with the root node. The edge exchange move is applied to the tree if the diameter bound is not violated.

For the two new heuristics, we applied the same analysis of the created backbones. For small instances, NSTC's backbones are similar to those of CBTC; STC is able to create backbones similar to an optimal backbone. For large instances, both heuristics are able to create low-weight backbones spanning large parts of the graph. The final experiments compared the performance of all construction heuristics for large Euclidean BDMST instances with up to $n = 1000$ nodes. Our two new heuristics STC and NSTC are able to significantly outperform the existing approaches. Especially for large problem instances, search performance increases by up to 27%.

In this paper, only Euclidean BDMST instances were analyzed. It might be interesting to test the performance of our new algorithms for instances with random distance weights. Another idea is to hybridize construction heuristics, for example, by applying STC to a solution created by another heuristic. In the future, we would like to design search operators for metaheuristics using the ideas presented in this paper. An analysis of backbones created by different crossover operators used in an evolutionary algorithm, will help to better understand the performance and bias of different approaches and will eventually lead to new, better performing operators.

References

Abdalla, A. and N. Deo (2002). "Random-tree diameter and the diameter-constrained MST". In: *Intern. Journal of Computer Mathematics* 79.6, pp. 651–663.

Achuthan, N., L. Caccetta, P. Caccetta, and J. Geelen (1994). "Computation methods for the diameter restricted minimum weight spanning tree problem". In: *Australasian Journal of Combinatorics* 10, pp. 51–71.

Beasley, J. E. (1990). "OR-Library: distributing test problems by electronic mail". In: *Journal of the Operational Research Society* 41.11, pp. 1069–1072.

Deo, N. and A. Abdalla (2000). "Computing a diameter-constrained minimum spanning tree in parallel". In: *Algorithms and Complexity*. Ed. by G. Bongiovanni, R. Petreschi, and G. Gambosi. Vol. 1767. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 17–31.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Gouveia, L. and T. L. Magnanti (2003). "Network flow models for designing diameter-constrained minimum-spanning and Steiner trees". In: *Networks* 41, pp. 159–173.

Gruber, M., J. van Hemert, and G. R. Raidl (2006). "Neighbourhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA, and ACO". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation, Seattle, Washington, USA*. GECCO '06. ACM, pp. 1187–1194.

Gruber, M. and G. R. Raidl (2005). "A new 0-1 ILP approach for the bounded diameter minimum spanning tree problem". In: *Proceedings of the 2nd International Network Optimization Conference*, pp. 178–185.

Gruber, M. and G. R. Raidl (2009). "Exploiting hierarchical clustering for finding bounded diameter minimum spanning trees on euclidean instances". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation, Montreal, QC, Canada*. GECCO '09. Montreal, Quebec, Canada: ACM, pp. 263–270.

Gruber, M. and G. R. Raidl (2010). "(Meta-)heuristic separation of jump cuts in a branch&cut approach for the bounded diameter minimum spanning tree problem". In: *Matheuristics*. Ed. by V. Maniezzo, T. Stuetzle, S. Voss, and R. Sharda. Vol. 10. Annals of Information Systems. Springer US, pp. 209–229.

Handler, G. Y. (1973). "Minimax location of a facility in an undirected tree graph". In: *Transportation Science* 7.3, pp. 287–293.

Julstrom, B. A. (2009). "Greedy heuristics for the bounded diameter minimum spanning tree problem". In: *Journal of Experimental Algorithmics (JEA)* 14, pp. 111–1114.

Kortsarz, G. and D. Peleg (1999). "Approximating the weight of shallow steiner trees". In: *Discrete Applied Mathematics* 93.2-3, pp. 265 –285.

Lucena, A., C. C. Ribeiro, and A. C. Santos (2009). "A hybrid heuristic for the diameter constrained minimum spanning tree problem". In: *Journal of Global Optimization* 46.3, pp. 363–381.

Prim, R. C. (1957). "Shortest connection networks and some generalizations". In: *The Bell System Technical Journal* 3, pp. 1389–1401.

Raidl, G. R. and M. Gruber (2008). "A Lagrangian relax-and-cut approach for the bounded diameter minimum spanning tree problem". In: *AIP Conference Proceedings* 1048.1. Ed. by T. E. Simos, G. Psihoyios, and C. Tsitouras, pp. 446–449.

Raidl, G. R. and B. A. Julstrom (2003b). "Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem". In: *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing, Melbourne, Florida*. New York, NY, USA: ACM, pp. 747–752.

Requejo, C. and E. Santos (2009). "Greedy heuristics for the diameter-constrained minimum spanning tree problem". In: *Journal of Mathematical Sciences* 161 (6), pp. 930–943.

Santos, A. dos, A. Lucena, and C. C. Ribeiro (2004). "Solving diameter constrained minimum spanning tree problems in dense graphs". In: *Experimental and Efficient Algorithms*, pp. 458–467.

Singh, A. and A. Gupta (2007). "Improved heuristics for the bounded-diameter minimum spanning tree problem". In: *Soft Computing – A Fusion of Foundations, Methodologies and Applications* 11.10, pp. 911–921.

Chapter 7

Summary & conclusions

The purpose of this thesis was to identify and analyze the properties of good and optimal solutions to constrained spanning tree problems as well as to show how the performance of heuristic optimization methods can be improved by considering problem-specific knowledge in the different components of heuristics and metaheuristics. This chapter summarizes the work and lists its major contributions.

7.1 Summary

Chapter 2 studies the optimal communication spanning tree (OCST) problem, which is a difficult and relevant combinatorial optimization problem. The chapter starts by investigating the edge orientation of optimal solutions compared to that of random, low-quality solutions. Edge orientation describes the relative orientation of an edge regarding the center of the graphs. The experimental analysis shows that edges in optimal solutions are not uniformly distributed, but edges pointing towards the center of the graph occur with higher frequency. Optimization methods for Euclidean OCST instances should therefore be biased towards such edges. We exploit this property by modifying the crossover operators of the direct tree representations edge-set and NetDir. These representations use crossover operators that iteratively select edges from two parents and transfer them to the offspring. Our extended crossover operators select edges to be included in the offspring based on edge weights *and* edge orientation. Low-weight parental edges that point towards the center are selected with a higher probability. These extended operators clearly outperform the existing approaches which only rely on edge weights. The high fit between the bias of the search operators and the properties of optimal solutions results in an efficient and robust search performance. The choice of the optimization approach (edge-set versus NetDir) is not that important.

In Chapter 3, the analysis of optimal solutions of the OCST problem is expanded. While the previous chapter analyzes the properties of edges contained in optimal solutions, this chapter analyzes the properties of the whole tree, such as the Wiener

index and node degrees. The Wiener index quantifies the degree of branching in a given tree. The results reveal that optimal solutions are structured in a star-like manner. There are few nodes with high node degrees and these nodes are located next to the graph's center. The majority of the nodes have very low node degrees. In particular, nodes with degree one are very common and located further away from the center. We exploit these insights to develop a construction heuristic that builds spanning trees with similar properties. Experimental results show that our construction heuristic is able to create solutions with similar quality to those of the more complicated and slower state-of-the-art heuristic from the literature. In a next step, we use the heuristic to seed the initial population of an evolutionary algorithm. An experimental study demonstrates the merits of using a biased initialization: the algorithm converges faster and finds better solutions in comparison to the same algorithm with a randomized initial population. This study again shows that creating metaheuristics with a bias that matches the properties of optimal solutions is a key element for designing successful optimization methods.

In Chapter 4, we use the previously acquired knowledge about high-quality solutions of OCST problems to design a biased fitness function. In contrast to other approaches that reward promising solution features by favoring edges with low weight and appropriate orientation, we use the guided local search (GLS) metaheuristic, which penalizes disadvantageous edges with large weights that do not point towards the center of the graph. We compare this approach to the evolutionary approaches from the previous chapters. Additionally, we propose a problem-specific variant of another state-of-the art crossover operator found in the literature. GLS is able to outperform the evolutionary approaches for larger OCST instances. However, we were not able to modify or increase GLS performance when we penalize either low-weight edges, edges with the wrong orientation, or both. Instead, GLS performance is independent of the type of penalization. This is in contrast to the classical assumption that considering more problem-specific knowledge about high-quality solutions increases search performance, as it was demonstrated with the evolutionary approaches. A possible explanation of this behavior is that a bias towards high-quality solution features has more influence on search behavior than a mechanism that just tries to avoid bad regions of the search space.

Chapter 5 focuses on a different $\mathcal{NP}$-hard combinatorial optimization problem, the quadratic minimum spanning tree (QMST) problem. The evaluation function of the QMST problem consists of two parts: the sum of the edge costs and the sum of the intercosts, which are the costs that arise for every pair of edges in a solution. Depending on the value-range of the edge costs and intercosts, the structure of high-quality solutions changes dramatically – which was ignored by previous publications on the problem. We change the cost function to be able to better analyze this fact. The ratio between the costs and intercosts determines

whether or not the QMST instance is more similar to the MST (high edge weights, low intercosts), and therefore easier to solve. In order to efficiently solve all kinds of QMST instances, the optimization method has to take into account the ratio of edge weights and intercosts. In a first step, we create construction heuristics for the QMST problem. Based on the insights gained with the heuristics, we develop problem-specific mutation and crossover operators in subsequent steps. Edges are selected depending on the edge weights and the intercosts that arise regarding the edges already contained in the solution. Hence, the operators favor low-weight edges that generate low intercosts. Using these biased search operators, we are able to create better solutions compared to unbiased approaches. The operators are used in an evolutionary algorithm and a simulated annealing approach, and we are able to significantly improve upon the state-of-the-art.

Chapter 6 investigates construction heuristics for the Euclidean bounded diameter spanning tree (BDMST) problem, which is another combinatorial optimization problem with applications in network design. The BDMST problem seeks a spanning tree with a constraint diameter. Depending on the diameter bound, the characteristic of the problem changes considerably. On the one hand, for large diameter bounds, optimal solutions are similar to the MST, and greedy approaches that favor low-weight edges perform best. On the other hand, for tight diameter bounds, these greedy approaches fail. The investigation of the backbones created by existing construction heuristics reveals their drawbacks: low performance results from the greediness which results in backbones spanning only a small area of the graph. The remaining nodes have to be connected using longer edges. While another existing approach is able to create a backbone spanning a larger area, the weight of the backbone is too high to create a better solution. Two new construction heuristics, called *savings tree construction* (STC) and *node selection tree construction* (NSTC), are proposed. STC starts from a star tree and iteratively improves the solution similarly to the savings heuristic used for vehicle routing problems. NSTC is an extension of existing greedy heuristics, but instead of choosing edges in a greedy manner or at random, edges are weighted by the costs and the cost savings for future edges that theses heuristics create. The edge weighting function estimates the tightness of the BDMST instance and is therefore suited for all types of instances (tight as well as loose bound). These new heuristics are able to create backbones that are more similar to optimal ones. This also enables them to significantly outperform the existing heuristics regarding the solution quality, in particular for large instances with tight diameter bounds. While this work only considers construction heuristics, the same analysis of backbones could be applied to crossover and mutation operators. It is likely that extending search operators in a similar way would lead to significant improvements in the search performance of metaheuristics for the BDMST problem.

7.2 Conclusions

The following paragraphs summarize the contributions of this thesis to the current state-of-the-art:

Design of problem-specific metaheuristics. When designing metaheuristic optimization methods, there is a trade-off between application range and effectiveness. For large real-world instances of combinatorial optimization problems out-of-the-box metaheuristics often fail, and optimization methods need to be adapted to the problem at hand. Knowledge about the structure of high-quality solutions can be exploited by introducing a bias into one of the components of the metaheuristic used. These problem-specific adaptations allow to increase search performance. This design task can be done systematically by first analyzing the characteristics of high-quality solutions and then exploiting this knowledge when implementing the metaheuristic. In this thesis, knowledge about the structure of high-quality solutions is generated by comparing the properties of optimal solutions to those of low-quality solutions. In order to conduct such a problem analysis, four essential components are needed: (a) a set of representative but solvable problem instances, (b) an exact or heuristic approach to generate optimal, or at least high-quality solutions for the test instances, (c) an approach to generate unbiased random solutions, and (d) a list of the relevant characteristics of solutions to the targeted problem. Using this framework allows developers of metaheuristics to identify the properties of high-quality solutions in a systematic way that can be used to generate high-performing optimization methods. It is also a good tool to provide a better explanation of experimental results on the performance of metaheuristics.

Relevant properties of constrained spanning tree problems. This thesis analyzes the characteristics of high-quality solutions for three different constrained spanning tree problems. We identify several relevant tree properties, that should be explored when analyzing a constrained spanning tree problem. Properties can be divided into two categories, one that considers the edges contained in a tree and one that considers the tree as a whole. The most obvious properties of the edges are those already considered by the evaluation function. For all three observed problems, these are the edge weights, since short edges usually lead to lower overall costs. For the QMST problem, the intercosts are also part of the evaluation function and are therefore relevant when analyzing the properties of high-quality solutions. Another important property is the edge orientation. Especially for problems with flows through the network, like the OCST problem, edges with low edge orientation help to create short paths. Structural properties describe the overall structure of a tree – for example if the tree is star-like or path-like. Relevant measurements

include the node degree distribution and the Wiener index. For some problems it might make sense to analyze only a subset of the tree, like it is done for the BDMST problem in this thesis, where the experiments only analyze the backbones.

Properties of high-quality solutions. For the OCST problem, this thesis shows that optimal solutions are not only biased towards the minimum spanning tree (MST), but there also is a higher probability that edges will be pointing towards the center of the graph. Therefore, optimization methods should be biased towards low-weight edges that point towards the center of the graph. To appropriately match the properties of high-quality solutions, the edge weights should have a higher influence than the orientation. Another study finds that high-quality solutions have a star-like structure and high-degree nodes are located near the tree's center. In contrast, nodes with lower node degree are located far away of the center. For optimal solutions, more than half of the nodes are leaves with a node degree of one. The average distance to the center decreases with increasing node degree.

High-quality solutions for the QMST problem are also similar to the MST. The ratio between edge weights and intercosts of a QMST instance tells us (a) if the edge weights have a higher influence and the instance is MST-like or (b) if the intercosts are of higher importance and the instance is more difficult to solve. Biased search operators should prefer low-weight edges that generate low intercosts regarding the edges contained in the solution.

For the BDMST problem the backbone of the tree has a huge influence on the solution quality. Lightweight backbones that span large areas also lead to high-quality solutions, since the remaining nodes can be connected via short edges. In contrast, if an approach selects the shortest edges too greedily, the backbone is lightweight but it only covers a small area. Then, the leaf nodes need to be connected with longer edges, which leads to higher overall costs. Hence, creating good backbones is the crucial challenge when creating optimization methods for BDMST problems, especially if the diameter bound is tight.

Problem-specific (meta-)heuristics for three constrained spanning tree problems. Finally, this work uses the knowledge on the structure of optimal solutions to create efficient and robust solution approaches for the OCST, QMST and BDMST problem. For the OCST problem, problem knowledge is integrated into the initialization method, the search operators and the fitness function. The developed construction heuristic yields solutions with a structure that is similar to that of optimal solutions; it is also simpler and faster than the best construction heuristic found in the literature. Biasing the initial population of an evolutionary algorithm creates a fast and robust solution method. Search operators use problem-specific knowledge by preferring edges with desired properties during the crossover. The

proposed evolutionary algorithm, using those biases crossover operators, is the current state-of-the-art for the OCST problem. A comparable performance is reached using a biased fitness function as shown with the guided local search approach.

For the QMST problem, biased initialization, mutation and crossover are designed in a problem-specific way. Using these operators in evolutionary algorithms and simulated annealing improves solution quality in comparison to existing unbiased operators. The resulting metaheuristics outperform the state-of-the-art approaches.

Successful construction heuristics for the BDMST problem generate lightweight backbones that still cover large areas. The proposed construction heuristics are designed to do exactly that. In comparison to existing heuristics, the bias towards low-weight edges is decreased. This enables them to create backbones that better match the optimal ones. Hence, the overall solution quality is significantly improved.

Bibliography

Abdalla, A. and N. Deo (2002). "Random-tree diameter and the diameter-constrained MST". In: *Intern. Journal of Computer Mathematics* 79.6, pp. 651–663.

Achuthan, N., L. Caccetta, P. Caccetta, and J. Geelen (1994). "Computation methods for the diameter restricted minimum weight spanning tree problem". In: *Australasian Journal of Combinatorics* 10, pp. 51–71.

Ahuja, R. K. and V. V. S. Murty (1987a). "Exact and heuristic algorithms for the optimum communication spanning tree problem". In: *Transportation Science* 21.3, pp. 163–170.

Ahuja, R. K. and V. V. S. Murty (1987b). "New lower planes for the network design problem". In: *Networks* 17.2, pp. 113–127.

Aringhieri, R., P. Hansen, and F. Malucelli (2001). "A linear algorithm for the hyper-wiener index of chemical trees". In: *Journal of Chemical Information and Computer Sciences* 41.4, pp. 958–963.

Assad, A. and W. Xu (1992). "The quadratic minimum spanning tree problem". In: *Naval Research Logistics* 39.3, pp. 399–417.

Beasley, J. E. (1990). "OR-Library: distributing test problems by electronic mail". In: *Journal of the Operational Research Society* 41.11, pp. 1069–1072.

Bern, M. W. (1988). "Two probabilistic results on rectilinear steiner trees". In: *Algorithmica* 3.2, pp. 191–204.

Blum, C. and A. Roli (2003). "Metaheuristics in combinatorial optimization: overview and conceptual comparison". In: *ACM Computing Surveys* 35.3, pp. 268–308.

Bonissone, P., R. Subbu, N. Eklund, and T. Kiehl (2006). "Evolutionary algorithms + domain knowledge = real-world evolutionary computation". In: *IEEE Transactions on Evolutionary Computation* 10.3, pp. 256–280.

Carrano, E. G., R. H. C. Takahashi, C. M. Fonseca, and O. M. Neto (2010). "Non-linear network optimization: an embedding vector space approach". In: *IEEE Transaction on Evolutionary Computation* 14 (2), pp. 206–226.

Chao, T.-H. and C. Hsu (1994). "Rectlinear steiner tree construction by local and global refinement". In: *IEEE Transactions on Computer-Aided Design of Integrated Circutits and Systems* 13.3, pp. 303–309.

Christofides, N. (1976). *Worst-case analysis of a new heuristic for the traveling salesman problem*. Tech. rep. 388. Pittsburgh, PA: Graduate School of Industratial Administration, Carnegie-Mellon University.

Contreras, I., E. Fernández, and A. Marín (2010). "Lagrangean bounds for the optimum communication spanning tree problem". In: *TOP* 18 (1). 10.1007/s11750-009-0112-5, pp. 140–157.

Deo, N. and A. Abdalla (2000). "Computing a diameter-constrained minimum spanning tree in parallel". In: *Algorithms and Complexity*. Ed. by G. Bongiovanni, R. Petreschi, and G. Gambosi. Vol. 1767. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 17–31.

Diestel, R. (2005). *Graph Theory*. 3rd ed. Graduate Texts in Mathematics 173. Heidelberg: Springer.

Droste, S. and D. Wiesmann (2003). "Advances in evolutionary computing". In: ed. by A. Ghosh and S. Tsutsui. New York, NY, USA: Springer-Verlag New York, Inc. Chap. On the design of problem-specific evolutionary algorithms, pp. 153–173.

Even, S. (1973). *Algorithmic Combinatorics*. New York: The Macmillan Company.

Fischer, T. and P. Merz (2007). "A memetic algorithm for the optimum communication spanning tree problem". In: *Hybrid Metaheuristics*. Ed. by T. Bartz-Beielstein, M. J. B. Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels. Vol. 4771. Springer, pp. 170–184.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Geisberger, R. and C. Vetter (2011). "Efficient routing in road networks with turn costs". In: *Proceedings of the 10th international conference on Experimental algorithms, Crete, Greece*. SEA'11. Berlin, Heidelberg: Springer-Verlag, pp. 100–111.

Gomory, R. E. and T. C. Hu (1961). "Multi-terminal network flows". In: *Journal of the Society for Industrial and Applied Mathematics* 9.4, pp. 551–570.

Gouveia, L. and T. L. Magnanti (2003). "Network flow models for designing diameter-constrained minimum-spanning and Steiner trees". In: *Networks* 41, pp. 159–173.

Gruber, M., J. van Hemert, and G. R. Raidl (2006). "Neighbourhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA, and ACO". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation, Seattle, Washington, USA*. GECCO '06. ACM, pp. 1187–1194.

Gruber, M. and G. R. Raidl (2005). "A new 0-1 ILP approach for the bounded diameter minimum spanning tree problem". In: *Proceedings of the 2nd International Network Optimization Conference*, pp. 178–185.

Gruber, M. and G. R. Raidl (2009). "Exploiting hierarchical clustering for finding bounded diameter minimum spanning trees on euclidean instances". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation, Montreal, QC, Canada*. GECCO '09. Montreal, Quebec, Canada: ACM, pp. 263–270.

Gruber, M. and G. R. Raidl (2010). "(Meta-)heuristic separation of jump cuts in a branch&cut approach for the bounded diameter minimum spanning tree problem". In: *Matheuristics*. Ed. by V. Maniezzo, T. Stuetzle, S. Voss, and R. Sharda. Vol. 10. Annals of Information Systems. Springer US, pp. 209–229.

Handler, G. Y. (1973). "Minimax location of a facility in an undirected tree graph". In: *Transportation Science* 7.3, pp. 287–293.

Hauschild, M. W., M. Pelikan, K. Sastry, and D. E. Goldberg (2012). "Using previous models to bias structural learning in the hierarchical boa". In: *Evolutionary Computation* 20.1, pp. 135–160.

Ho, J.-M., G. Vijayan, and C. K. Wong (1990). "New algorithms for the rectilinear steiner tree problem". In: *IEEE Transactions on Computer-Aided Design* 9, pp. 185–193.

Hu, T. C. (1974). "Optimum communication spanning trees." In: *SIAM Journal on Computing* 3.3, pp. 188–195.

Hwang, F. K. (1976). "On steiner minimal trees with rectilinear distance". In: *SIAM J. Applied Math.* 1, pp. 104–114.

Julstrom, B. A. (2009). "Greedy heuristics for the bounded diameter minimum spanning tree problem". In: *Journal of Experimental Algorithmics (JEA)* 14, pp. 111–1114.

Julstrom, B. A. and G. R. Raidl (2001). "Weight-biased edge-crossover in evolutionary algorithms for two graph problems". In: *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing, Las Vegas, Nevada, United States*. New York, NY, USA: ACM, pp. 321–326.

Julstrom, B. A. and G. R. Raidl (2002). "Initialization is robust in evolutionary algorithms that encode spanning trees as sets of edges". In: *SAC '02: Proceedings of the 2002 ACM symposium on applied computing, Madrid, Spain*. New York, NY, USA: ACM, pp. 547–552.

Karaboga, D. and B. Basturk (2007). "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm". In: *Journal of Global Optimization* 39 (3), pp. 459–471.

Kershenbaum, A. (1993). *Telecommunications network design algorithms*. New York: McGraw Hill.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). "Optimization by simulated annealing". In: *Science* 220, pp. 671–680.

Kortsarz, G. and D. Peleg (1999). "Approximating the weight of shallow steiner trees". In: *Discrete Applied Mathematics* 93.2-3, pp. 265 –285.

Li (1992). "Random texts exhibit Zipf's law-like word frequency distribution". In: *IEEE Transactions on Information Theory* 38.

Li, Y. and Y. Bouchebaba (2000). "A new genetic algorithm for the optimal communication spanning tree problem". In: *AE '99: Selected Papers from the 4th European Conference on Artificial Evolution*. London, UK: Springer-Verlag, pp. 162–173.

Lucena, A., C. C. Ribeiro, and A. C. Santos (2009). "A hybrid heuristic for the diameter constrained minimum spanning tree problem". In: *Journal of Global Optimization* 46.3, pp. 363–381.

Michalewicz, Z. and D. B. Fogel (2004). *How to Solve It: Modern Heuristics*. 2nd. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Öncan, T. and A. P. Punnen (2010). "The quadratic minimum spanning tree problem: a lower bounding procedure and an efficient search algorithm". In: *Computers and Operations Research* 37.10, pp. 1762–1773.

Palmer, C. and A. Kershenbaum (1995). "An approach to a problem in network design using genetic algorithms". In: *Networks* 26, pp. 151–163.

Palmer, C. C. (1994). "An approach to a problem in network design using genetic algorithms". PhD thesis. Troy, NY, USA: Polytechnic University.

Palubeckis, G., D. Rubliauskas, and A. Targamadze (2010). "Metaheuristic Approaches for the Quadratic Minimum Spanning Tree Problem". In: *Information Technology and Control* 39.4, pp. 257–268.

Papadimitriou, C. H. and M. Yannakakis (1991). "Optimization, approximation, and complexity classes". In: *Journal of Computer and System Sciences* 43.3, pp. 425–440.

Papadimitriou, C. and M. Yannakakis (1988). "Optimization, approximation, and complexity classes". In: *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing, Chicago, Illinois, United States*. New York, NY, USA: ACM Press, pp. 229–234.

Paulden, T. J. (2007). "Combinatorial spanning tree representations for evolutionary algorithms". PhD thesis. Department of Computer Science, University of Exeter, UK.

Peleg, D. and E. Reshef (1998). "Deterministic polylog approximation for minimum communication spanning trees". In: *ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming*. London, UK: Springer-Verlag, pp. 670–681.

Poosala, V. (1995). *Zipf's Law*. Tech. rep. University of Wisconsin.

Prim, R. C. (1957). "Shortest connection networks and some generalizations". In: *The Bell System Technical Journal* 3, pp. 1389–1401.

Prüfer, H. (1918). "Neuer Beweis eines Satzes über Permutationen". In: *Archiv für Mathematik und Physik* 27, pp. 742–744.

Raidl, G. R. (2000). "An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem". In: *Proc. of the 2000 Congress on Evolutionary Computation*. Piscataway, NJ: IEEE Service Center, pp. 104–111.

Raidl, G. R. and J. Gottlieb (2005). "Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: a case study for the multidimensional knapsack problem". In: *Evolutionary Computation* 13.4, pp. 441–475.

Raidl, G. R. and M. Gruber (2008). "A Lagrangian relax-and-cut approach for the bounded diameter minimum spanning tree problem". In: *AIP Conference Proceedings* 1048.1. Ed. by T. E. Simos, G. Psihoyios, and C. Tsitouras, pp. 446–449.

Raidl, G. R. and B. A. Julstrom (2003a). "Edge sets: an effective evolutionary coding of spanning trees". In: *IEEE Transactions on Evolutionary Computation* 7.3, pp. 225–239.

Raidl, G. R. and B. A. Julstrom (2003b). "Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem". In: *SAC*

'03: Proceedings of the 2003 ACM symposium on Applied computing, Melbourne, Florida. New York, NY, USA: ACM, pp. 747–752.

Requejo, C. and E. Santos (2009). "Greedy heuristics for the diameter-constrained minimum spanning tree problem". In: *Journal of Mathematical Sciences* 161 (6), pp. 930–943.

Reshef, E. (1999). "Approximating minimum communication cost spanning trees and related problems". MA thesis. Rehovot 76100, Israel: Feinberg Graduate School of the Weizmann Institute of Science.

Richards, D. (1989). "Fast heuristic algorithms for rectilinear steiner trees". In: *Algorithmica* 4, pp. 191–207.

Robins, G. and A. Zelikovsky (2005). "Tighter bounds for graph steiner tree approximation". In: *SIAM J. Discret. Math.* 19.1, pp. 122–134.

Robins, G. and J. Salowe (1994). "On the maximum degree of minimum spanning trees". In: *Proceedings of the tenth annual symposium on Computational geometry*. ACM New York, NY, USA, pp. 250–258.

Rothlauf, F. (2007). *Design and Application of Metaheuristics*. Habilitationsschrift. Department of Information Systems 1, University of Mannheim, Germany.

Rothlauf, F. and D. E. Goldberg (1999). "Tree network design with genetic algorithms - an investigation in the locality of the prüfernumber encoding". In: *Late Breaking Papers at the Genetic and Evolutionary Computation Conference 1999*. Ed. by S. Brave and A. S. Wu. Orlando, Florida, USA: Omni Press, pp. 238–244.

Rothlauf, F. (2006). *Representations for genetic and evolutionary algorithms (2. ed.)* Springer, pp. I–XVII, 1–325.

Rothlauf, F. (2009a). "An encoding in metaheuristics for the minimum communication spanning tree problem." In: *INFORMS Journal on Computing* 21.4, pp. 575–584.

Rothlauf, F. (2009b). "On Optimal Solutions for the Optimal Communication Spanning Tree Problem". In: *Operations Research* 57.2, pp. 413–425.

Rothlauf, F. (2009c). "On the Bias and Performance of the Edge-Set Encoding". In: *IEEE Transactions on Evolutionary Computation* 13.3, pp. 486–499.

Rothlauf, F. (2011). *Design of Modern Heuristics: Principles and Application*. 1st. Springer.

Rothlauf, F., D. E. Goldberg, and A. Heinzl (2002). "Network random keys: a tree representation scheme for genetic and evolutionary algorithms". In: *Evolutionary Computaton* 10.1, pp. 75–97.

Santos, A. dos, A. Lucena, and C. C. Ribeiro (2004). "Solving diameter constrained minimum spanning tree problems in dense graphs". In: *Experimental and Efficient Algorithms*, pp. 458–467.

Sharma, P. (2006). "Algorithms for the optimum communication spanning tree problem". In: *Annals of Operations Research* 143.1, pp. 203–209.

Singh, A. and A. Gupta (2007). "Improved heuristics for the bounded-diameter minimum spanning tree problem". In: *Soft Computing – A Fusion of Foundations, Methodologies and Applications* 11.10, pp. 911–921.

Soak, S.-M. (2006). "A new evolutionary approach for the optimal communication spanning tree problem". In: *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences* E89-A.10, pp. 2882–2893.

Soak, S.-M., D. W. Corne, and B.-H. Ahn (2006). "The edge-window-decoder representation for tree-based problems". In: *IEEE Transactions on Evolutionary Computation* 10.2, pp. 124–144.

Steitz, W. and F. Rothlauf (2008). "Orientation matters: how to efficiently solve OCST problems with problem-specific EAs". In: *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, Atlanta, GA, USA*. New York, NY, USA: ACM, pp. 563–570.

Steitz, W. and F. Rothlauf (2012). "Edge orientation and the design of problem-specific crossover operators for the ocst problem." In: *IEEE Transactions on Evolutionary Computation* 16.1, pp. 108–116.

Sundar, S. and A. Singh (2010). "A swarm intelligence approach to the quadratic minimum spanning tree problem". In: *Information Sciences* 180.17, pp. 3182–3191.

Surry, P. D. and N. J. Radcliffe (1996). "Inoculation to initialise evolutionary search". In: *Selected Papers from AISB Workshop on Evolutionary Computing*. London, UK, UK: Springer-Verlag, pp. 269–285.

Tsang, E., P. Mills, and J. Ford (2002). *Extending Guided Local Search - Towards a Metaheuristic Algorithm With No Parameters To Tune*. Tech. rep. 371. Department of Computer Science, University of Essex.

Voudouris, C. (1997). "Guided Local search for combinatorial optimization problems". PhD thesis. Colchester,UK: Department of computer science, University of Essex.

Voudouris, C. (2003). "Guided local sarch". In: *Handbook of Metaheuristics*. Ed. by F. Glover and G. A. Kochenberger. Springer, pp. 185–218.

Voudouris, C. and E. Tsang (1999). "Guided local search and its application to the traveling salesman problem". In: *European Journal of Operational Research* 113.2, pp. 469–499.

Winter, S. (2002). "Modeling costs of turns in route planning". In: *GeoInformatica* 6 (4), pp. 345–361.

Wolpert, D. H. and W. G. Macready (1997). "No free lunch theorems for optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1, pp. 67–82.

Wu, B. Y. and K.-M. Chao (2004). *Spanning Trees and Optimization Problems*. CRC Press.

Wu, B. Y., K.-M. Chao, and C. Y. Tang (2000a). "A polynomial time approximation scheme for optimal product-requirement communication spanning trees". In: *Journal of Algorithms* 36.2, pp. 182–204.

Wu, B. Y., K.-M. Chao, and C. Y. Tang (2000b). "Approximation algorithms for some optimum communication spanning tree problems". In: *Discrete Applied Mathematics* 102.3, pp. 245–266.

Zhou, G. and M. Gen (1998). "An effective genetic algorithm approach to the quadratic minimum spanning tree problem". In: *Computers and Operations Research* 25 (3), pp. 229–237.

Zipf, G. K. (1949). *Human Behaviour and the Principle of Least Effort: an Introduction to Human Ecology*. Addison-Wesley.