OPERATIONS RESEARCH
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0030-364X | EISSN 1526-5463 | 00 | 0000 | 0001

INFORMS
DOI 10.1287/xxxx.0000.0000

On Optimal Solutions for the Optimal Communication Spanning Tree Problem

Franz Rothlauf
University of Mainz, Dept. of Information System, 55099 Mainz, Germany, rothlauf@uni-mainz.de,
http://www.wi.bwl.uni-mainz.de/rothlauf

This paper presents an experimental investigation into the properties of the optimal communication spanning tree (OCST) problem. The OCST problem seeks a spanning tree that connects all the nodes and satisfies their communication requirements at a minimum total cost. The paper compares the properties of random trees to the properties of the best solutions for the OCST problem that are found using an evolutionary algorithm. The results show on average, that the optimal solution and the minimum spanning tree (MST) share a higher number of links than the optimal solution and a random tree. Furthermore, optimal solutions for OCST problems with randomly chosen distance weights share a higher number of links with an MST than OCST problems with Euclidean distance weights. This intuitive similarity between optimal solutions and MSTs suggests that some heuristic optimization methods for OCST problems might be improved by starting with an MST. Using an MST as starting solution for a greedy search in the tested cases either improves median running time up to a factor of 10 while finding solutions of the same quality, or increases solution quality up to a factor of 100 while using the same number of search steps in comparison to starting the greedy search from a random tree. Starting a local search, a simulated annealing approach, and a genetic algorithm from an MST increases solution quality up to a factor of three in comparison to starting from a random solution.

Subject classifications: Networks/graphs: Tree algorithms. Communications: Topological network design. Simulation: Statistical analysis
Area of review: Telecommunications and Networking
History:

1. Introduction

The optimal communication spanning tree (OCST) problem [19] seeks a spanning tree that connects all given nodes and satisfies their communication requirements at a minimum total cost. Like other constrained spanning tree problems, the OCST problem is $\mathcal{NP}$-hard [13, ND7]. This paper presents an investigation into the properties of optimal solutions for OCST problems. It examines the properties of existing OCST problem instances from the literature, as well as randomly generated OCST problems using either Euclidean, or randomly chosen distance weights.

The main results presented in this paper are:

1. For OCST problems, on average, the distance between optimal solutions and the minimum spanning tree (MST) is lower than the distance between optimal solutions and randomly created trees. Furthermore, on average, MSTs share a higher number of edges with optimal solutions than with randomly created solutions.

2. Optimal solutions for OCST problems with randomly chosen distance weights have, on average, a lower distance to an MST than optimal solutions for OCST problems with Euclidean distance weights.

3. The intuitive similarity between MSTs and optimal solutions suggests that some heuristic optimization methods for the OCST problem might be improved by starting with an MST. In particular, this is shown for a simple greedy strategy. Starting from an MST either improves median running time up to a factor of 10 while finding solutions of the same quality or increases solution quality up to a factor of 100 while using the same number of search steps as when starting from a random tree.

4. Starting a local search, a simulated annealing approach, and a genetic algorithm from an MST increases solution quality up to a factor of three in comparison to starting from a random tree. In comparison to existing state-of-the-art approaches solution quality increases up to 10%. Furthermore, starting from an MST increases solution quality up to a factor of two in comparison to starting from solutions generated by an $O(\log^3 n)$ approximation (random distance weights), and up to 10% in comparison to starting from solutions generated by an $O(\log n)$ approximation (Euclidean distance weights).

The following section describes the OCST problem and analyzes existing problem instances from the literature. Section 3 analyzes random problem instances. Section 4 discusses the impact of the results on the design of optimization methods, especially search heuristics. In particular, the performance of a greedy search strategy and other heuristic optimization methods starting from an MST is investigated in Section 5.

2. The Optimal Communication Spanning Tree Problem

2.1. Problem Description

The OCST problem [19] seeks a tree that connects all given nodes and satisfies their communication requirements for a minimum total cost. The number and positions of the nodes are given a priori and the cost of the tree is determined by the cost of the edges. An edge's flow is the sum of the communication demands between all pairs of nodes communicating either directly, or indirectly, over the edge. Therefore, the cost for each edge is not fixed a priori but depends on the structure of the tree.

Let $G = (V, E)$ be a connected, undirected graph with $n = |V|$ nodes and $m = |E|$ edges. There are communication, or transportation demands, between every pair of nodes. An $n \times n$ *demand matrix* $R = (r_{ij})$ specifies the demands, where r_{ij} is the amount of traffic required between nodes v_i and v_j. Similarly, an $n \times n$ *distance matrix* $W = w_{ij}$ determines the distance weights between each pair of nodes. A tree $T = (V, F)$ where $F \subseteq E$ and $|F| = |V| - 1$ is called a *spanning tree* of G if it connects all the nodes. The weight $w(T)$ of the spanning tree is the weighted sum over all pairs of vertices of the cost of the path between the pair in T. The communication cost over the tree T is defined as

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

where $B = b_{ij}$ denotes the traffic flowing directly and indirectly across the edge connecting nodes i and j. It is calculated according to the structure of T. T is the OCST if $w(T) \leq w(T')$ for all other spanning trees T'. The OCST problem becomes the MST problem if $w(T) = \sum_{i,j \in V} w_{ij}$. Then, T is the (simple) MST if $w(T) \leq w(T')$ for all other spanning trees T'.

Cayley's formula identifies the number of spanning trees on n nodes as n^{n-2} [7]. Furthermore, there are n different stars on a graph of n nodes. The similarity between two spanning trees T_i and T_j can be measured using the distance $d_{ij} \in \{0, 1, \ldots, n-1\}$ as

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

where l^i_{uv} is 1 if an edge from u to v exists in T_i and 0 if it does not exist in T_i. The number of edges that two trees T_i and T_j have in common is $n - 1 - d_{ij}$.

Like other constrained spanning tree problems, the OCST problem is $\mathcal{NP}$-hard [13, p. 207]. Furthermore, it was later shown in Reshef [38] that the problem is $\mathcal{MAX}$ $\mathcal{SNP}$-hard [31] which means it can have a constant-factor approximation algorithm, but no approximation schemes unless $\mathcal{P} = \mathcal{NP}$. Therefore, the OCST problem belongs to the class of optimization problems that behave like MAX-3SAT [13].

The current best approximation algorithm for the OCST problem with Euclidean distance weights finds a tree with cost $w(T) = O(\log n) \cdot w(G)$ [32, 38], where $w(G) = \sum_{i,j \in E} r_{ij} t^G_{ij}$ is the volume of communication in the complete graph G and t^G_{ij} is the sum of all weights along the shortest path between i and j in G. $w(G)$ is a trivial lower bound for $w(T)$ as the weight of a path between node i and j in a spanning tree T is greater, or equal to, the weight of the path in G. Furthermore, due to the $\mathcal{NP}$-hardness of the problem many researchers have applied search heuristics to OCST problems. In this context a large variety of representations and heuristic search operators have been developed [30, 4, 3, 28, 27, 33, 25, 9, 41]. Raidl and Julstrom [36] presented a genetic algorithm using the edge-set encoding which represents the current state-of-the-art heuristic for OCST problems.

2.2. Analysis of Existing Problem Instances

Test instances for the OCST problem have been proposed in the literature by Palmer [30], Berry et al. [3], and Raidl [37]. Palmer [30] described OCST problems with six (palm6), twelve (palm12), 24 (palm24), 47, and 98 nodes. The nodes correspond to cities in the United States and the distances between the nodes are obtained from a tariff database. The inter-node traffic demands are inversely proportional to the distances between the nodes. Berry et al. [3] presented three instances of the OCST problem, one with six nodes (berry6) and two with 35 nodes (berry35 and berry35u). For berry35u, $w_{ij} = 1$. The cost of the optimal solution for berry6 is 534 and for berry35 is 16,915 [28]. Finally, Raidl [37] proposed several test instances ranging from 10 to 100 nodes. The distances and the traffic demands were generated randomly and are uniformly distributed in $[0, 100]$.

In the following paragraphs, we perform an investigation into specific properties of these test problems. The goal is to gain additional information about the problems as well as their optimal solutions. To identify relevant properties of randomly generated solutions, we randomly generated 10,000 solutions (trees) for each test problem. Raidl and Julstrom [36] showed that it is not as simple as it might seem to choose spanning trees of a graph so that all are equally likely, as techniques based on Prim's or Kruskal's MST algorithms using randomly chosen distance weights, do not associate uniform probabilities with spanning trees. An appropriate method for generating unbiased trees is generating random Prüfer numbers [35, 29] and creating the corresponding tree from the Prüfer number. Prüfer numbers belong to the class of Cayley codes and are a constructive proof of Cayley's theorem [7]. They describe a pair of inverse one-to-one mappings between spanning trees on n nodes and strings of length $n-2$ using an alphabet of cardinality n. The Prüfer number string corresponding to a tree is generated by sequentially deleting the tree's leaves and recording the neighbors of these leaves. Different Cayley codes use a different set of rules that determine the order of leaf deletions. A tree can be constructed from a randomly chosen Prüfer number in $O(n \log n)$. The use of Prüfer numbers ensures that the probability of generating trees is uniformly distributed.

Table 1 lists the properties of randomly created solutions and the properties of the optimal (or best known solution) for the test instances. It shows the mean μ, the normalized mean μ/n, and the standard deviation σ of the distances $d_{mst,rnd}$ between randomly generated trees and MSTs and of the minimum distances $\min(d_{star,rnd})$ between random trees and the n different stars. In the instance berry35u, all distance weights are uniform ($w_{ij} = 1$), so all spanning trees are minimal and for each $d_{mst,rnd} = 0$. For the optimal solutions, we calculated their distance to an MST ($d_{mst,opt}$ and $d_{mst,opt}/n$), their minimum distance towards a star ($\min(d_{star,opt})$ and $\min(d_{star,opt})/n$), and the mean distance $d_{opt,rnd}$ between the optimal solution and random trees. Furthermore, we show the cost $w(T^{best})$ of the optimal solution. Comparing $d_{mst,opt}$ with $d_{opt,rnd}$ and $d_{mst,rnd}$ reveals that for all test instances, $d_{mst,opt} < \mu(d_{opt,rnd})$ and $d_{mst,opt} < \mu(d_{mst,rnd})$. This means that, on average, the best solutions share more links with MSTs than with randomly generated solutions. Comparing $\min(d_{star,opt})$ and $\mu(\min(d_{star,rnd}))$ does not reveal any differences. Randomly created solutions

Table 1 Properties of randomly created solutions and optimal solutions for the test instances

		Properties of random solutions				Properties of optimal solutions			
problem		$d_{mst,rnd}$		$\min(d_{star,rnd})$		$d_{mst,opt}$	$\min(d_{star,opt})$	$d_{opt,rnd}$	$w(T^{best})$
instance	n	μ (μ/n)	σ	μ (μ/n)	σ	$\left(\frac{d_{mst,opt}}{n}\right)$	$\left(\frac{\min(d_{star,opt})}{n}\right)$	μ	
palm6	6	3.36 (0.56)	0.91	2.04 (0.34)	0.61	1 (0.17)	2 (0.33)	3.40	693,180
palm12	12	9.17 (0.76)	1.17	7.22 (0.60)	0.75	5 (0.42)	7 (0.58)	9.05	3,428,509
palm24	24	21.05 (0.88)	1.30	18.50 (0.77)	0.80	12 (0.50)	17 (0.71)	21.07	1,086,656
raidl10	10	7.20 (0.72)	1.10	5.42 (0.54)	0.70	3 (0.30)	4 (0.40)	7.08	53,674
raidl20	20	17.07 (0.85)	1.27	14.69 (0.73)	0.77	4 (0.20)	14 (0.70)	17.10	157,570
raidl50	50	47.09 (0.94)	1.32	43.88 (0.88)	0.87	13 (0.26)	41 (0.82)	47.10	806,864
raidl75	75	72.02 (0.96)	1.36	68.55 (0.91)	0.83	18 (0.24)	68 (0.91)	72.03	1,717,491
raidl100	100	97.09 (0.97)	1.36	93.29 (0.93)	0.89	32 (0.32)	90 (0.90)	97.05	2,561,543
berry6	6	3.51 (0.59)	0.83	2.03 (0.34)	0.61	0 (0)	2 (0.33)	3.51	534
berry35u	35	-	-	29.19 (0.83)	0.83	-	28 (0.80)	32.01	16,273
berry35	35	32.05 (0.92)	1.32	29.16 (0.83)	0.83	0 (0)	30(0.86)	32.05	16,915

have approximately the same expected minimum distance to a star as does the optimal solution. Furthermore, $\mu(d_{opt,rnd})$ is approximately equal to $\mu(d_{mst,rnd})$.

3. Analysis of Random Problem Instances

We created for each problem size, ranging from eight to 28 nodes, 100 random problem instances. The real-valued demands r_{ij} are generated randomly and are uniformly distributed in $]0, 100]$. For the w_{ij}, there are two possibilities:

- Random distance weights: The real-valued distance weights w_{ij} are generated randomly and are uniformly distributed in $]0, 100]$.
- Euclidean distance weights: The nodes are randomly placed on a 1,000x1,000 grid. The w_{ij} are the Euclidean distances between the nodes i and j.

For each of the 100 problem instances, we generated 10,000 random trees. Table 2 presents the mean μ, the normalized mean μ/n, and the standard deviation σ of $d_{mst,rnd}$ and $\min(d_{star,rnd})$. As we get the same results using random or Euclidean distance weights, we neglect the distance weights used. Both $\mu(d_{mst,rnd})$ and $\mu(\min(d_{star,rnd}))$, increase approximately linearly with n.

For finding optimal or near-optimal solutions for randomly created problem instances we implemented an evolutionary algorithm (EA). Although EAs are heuristic search methods that can not guarantee that the optimal solution is found, we choose its design in such a way that we can assume that the found solution is optimal or near-optimal. Due to the proposed EA design and due to the $\mathcal{NP}$-hardness of the problem, the effort for finding optimal solutions using an EA is high, and optimal solutions can only be determined for small problem instances ($n < 30$). Instead of an EA, in theory, other heuristic and non-heuristic optimization approaches like integer programming can also be used for determining optimal solutions. For example, the OCST problem can be represented as an integer linear program and the relaxed linear program can be solved exactly in polynomial time. The cost of the found (fractional) solution is a lower bound on the cost of the optimal tree. However, to get a valid solution the fractional solution must be rounded into an integral feasible solution. There are no methods available that can perform the rounding in polynomial time and the ratio between the cost of the optimal integral solution and the optimal fractional solution is larger than $\Omega(\log n)$ [38].

Like other search approaches that are applied to $\mathcal{NP}$-hard problems like the OCST problem, the computational effort of EAs grows exponentially with n. For a fixed n the probability that an EA fails to find an optimal solution is $O(\exp(-N))$ [17], where N is the EA's population size. As EA performance increases with N, we iteratively apply an EA to the problem and increase N after

Table 2 Properties of randomly created solutions for the OCST problem

	$d_{mst,rnd}$		$\min(d_{star,rnd})$	
nodes	μ (μ/n)	σ	μ (μ/n)	σ
8	5.25 (0.66)	1.04	3.74 (0.47)	0.62
10	7.19 (0.72)	1.11	5.50 (0.55)	0.67
12	9.17 (0.77)	1.16	7.31 (0.61)	0.72
14	11.14 (0.80)	1.20	9.15 (0.65)	0.75
16	13.13 (0.82)	1.22	11.00 (0.69)	0.76
18	15.11 (0.84)	1.24	12.88 (0.72)	0.77
20	17.10 (0.86)	1.26	14.78 (0.74)	0.77
22	19.09 (0.87)	1.27	16.69 (0.76)	0.77
24	21.09 (0.88)	1.29	18.60 (0.78)	0.80
26	23.06 (0.89)	1.30	20.44 (0.79)	0.80
28	25.05 (0.895)	1.30	22.70 (0.81)	0.72

n_{iter} runs. We start by applying an EA n_{iter} times to an OCST problem using a population size of N_0. T_0^{best} denotes the best solution of cost $w(T_0^{best})$ that is found during the n_{iter} runs. In a next round, we double the population size and again apply an EA n_{iter} times with a population size of $N_1 = 2N_0$. T_1^{best} denotes the best solution with cost $w(T_1^{best})$ that can be found in the second round. 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. Although the computational effort of this approach is high and grows exponentially with n, it allows us to find optimal (or near-optimal) solutions for OCST problems with low n.

For determining the optimal solution we use a classical canonical genetic algorithm (cGA) [16] with crossover and mutation and traditional parameter settings. The problem was encoded using the NetKey encoding [43]. This representation ensures good EA performance (similar to the edge-sets from Raidl and Julstrom [36]) [41] and represents all possible trees uniformly. The cGA uses one-point 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 $p_{mut} = 0.02$.

In our experiments, we applied the cGA for each problem size n to the same 100 problem instances as in Table 2. The purpose is to find optimal or near-optimal solutions using a GA and to examine the properties of the best found solutions T_i^{best}. We started with $N_0 = 100$ and set $n_{iter} = 20$. The computational effort for the experiments is high. For calculating the optimal or near-optimal solutions of all 100 problem instances for $n = 28$ using the proposed GA, we spent some 100 hours of computing time on a P4 processor at 2 GHz.

Table 3 and 4 present the properties of the best solutions T_i^{best} that have been found for the 100 problem instances of each problem size n using the proposed cGA. We distinguish between Euclidean w_{ij} (Table 3) and random w_{ij} (Table 4). The tables show for different n the mean μ, the normalized mean μ/n, and the standard deviation σ of $d_{mst,opt}$, $\min(d_{star,opt})$, and $d_{opt,rnd}$, the average population size N_i in the last GA round i, and the mean μ and standard deviation σ of the cost $w(T_i^{best})$ of the best found solution T_i^{best}.

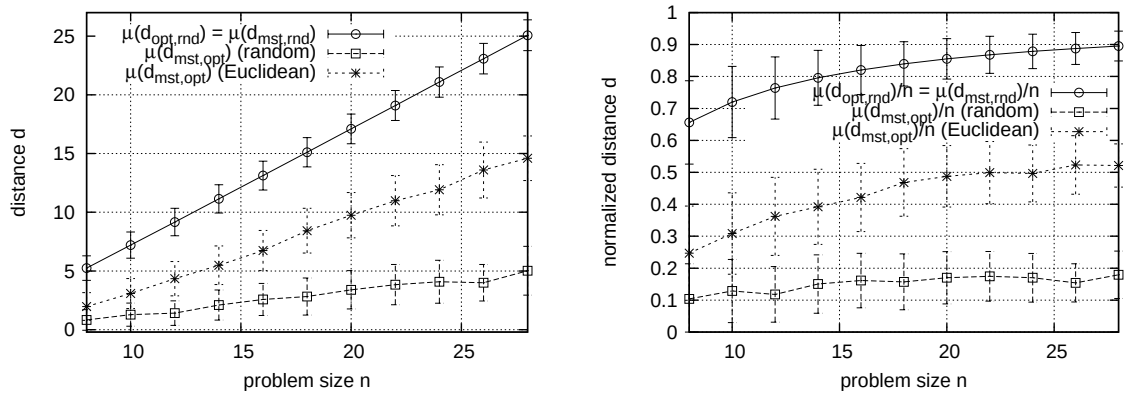
For random w_{ij}, the average distance $d_{mst,opt}$ between optimal solutions and MSTs is lower than for Euclidean w_{ij}. Comparing the properties of the best found solutions to the properties of randomly created solutions (see also Table 2) reveals that $\mu(d_{mst,opt}) < \mu(d_{opt,rnd})$ and $\mu(d_{mst,opt}) < \mu(d_{mst,rnd})$. This means, on average, optimal solutions share more links with MSTs than with random solutions. Furthermore, the distances $d_{opt,rnd}$ are similar to $d_{mst,rnd}$ and $\min(d_{star,opt})$ is slightly lower than $\min(d_{star,rnd})$, especially for Euclidean w_{ij}. However, this effect is weak and can be neglected in comparison to the low distance $d_{mst,opt}$.

Table 3 Properties of optimal solutions for OCST problems with Euclidean distance weights

	$d_{mst,opt}$		$\min(d_{star,opt})$		$d_{opt,rnd}$		N_i	$w(T_i^{best})$	
nodes	μ (μ/n)	σ	μ (μ/n)	σ	μ (μ/n)	σ	μ	μ	σ
8	1.98 (0.25)	1.18	2.96 (0.37)	0.93	5.25 (0.66)	0.01	389	899,438	177,140
10	3.09 (0.31)	1.28	4.46 (0.45)	0.98	7.20 (0.72)	0.01	1,603	1,496,052	258,705
12	4.35 (0.36)	1.47	5.88 (0.49)	1.20	9.17 (0.76)	0.01	1,773	2,204,858	301,507
14	5.49 (0.39)	1.65	7.67 (0.55)	1.22	11.14 (0.80)	0.01	3,408	3,081,755	373,367
16	6.74 (0.42)	1.71	9.28 (0.58)	1.24	13.12 (0.82)	0.01	7,048	4,104,579	463,588
18	8.43 (0.47)	1.90	10.87 (0.60)	1.26	15.11 (0.84)	0.01	12,504	5,210,432	528,976
20	9.75 (0.49)	1.92	12.68 (0.63)	1.15	17.10 (0.86)	0.01	20,232	6,625,619	649,722
22	10.98 (0.50)	2.14	14.48 (0.66)	1.51	19.09 (0.87)	0.02	28,312	7,811,178	795,036
24	11.92 (0.50)	2.14	16.48 (0.69)	1.23	21.08 (0.88)	0.01	40,784	9,320,862	718,694
26	13.60 (0.52)	2.38	18.28 (0.70)	1.62	23.08 (0.89)	0.01	67,673	11,454,844	858,822
28	14.60 (0.52)	1.90	19.70 (0.70)	1.42	25.07 (0.89)	0.01	98,467	13,121,110	723,003

Table 4 Properties of optimal solutions for OCST problems with random distance weights

	$d_{mst,opt}$		$\min(d_{star,opt})$		$d_{opt,rnd}$		N_i	$w(T_i^{best})$	
nodes	μ (μ/n)	σ	μ (μ/n)	σ	μ (μ/n)	σ	μ	μ	σ
8	0.83 (0.10)	0.88	3.42 (0.43)	0.68	5.25 (0.66)	0.01	234	50,807	18,999
10	1.28 (0.13)	0.99	4.95 (0.50)	0.81	7.20 (0.72)	0.01	381	68,638	22,568
12	1.42 (0.12)	1.04	6.87 (0.57)	0.75	9.16 (0.76)	0.01	478	90,842	28,207
14	2.10 (0.15)	1.28	8.52 (0.61)	0.97	11.14 (0.80)	0.01	1,290	117,011	37,151
16	2.58 (0.16)	1.36	10.23 (0.64)	0.95	13.12 (0.82)	0.01	2,208	136,275	36,437
18	2.83 (0.16)	1.58	12.00 (0.67)	1.13	15.11 (0.84)	0.01	3,336	162,975	48,259
20	3.40 (0.17)	1.63	13.94 (0.70)	1.10	17.10 (0.86)	0.01	6,512	183,367	49,179
22	3.84 (0.17)	1.71	15.93 (0.72)	0.97	19.01 (0.86)	0.01	7,808	208,033	52,091
24	4.08 (0.17)	1.83	17.76 (0.74)	1.04	21.10 (0.88)	0.01	10,432	228,862	58,333
26	4.60 (0.18)	1.56	19.50 (0.75)	0.85	23.07 (0.89)	0.01	15,250	248,768	53,088
28	5.02 (0.18)	2.07	21.96 (0.78)	0.83	25.06 (0.90)	0.01	19,456	271,897	70,205



(a)distances d over n (b)normalized distances d/n over n

Figure 1 We show how $\mu(d_{opt,rnd})$ and $\mu(d_{mst,opt})$ using random and Euclidean distance weights depend on the problem size n. The error bars indicate the standard deviation. As $\mu(d_{mst,opt}) < \mu(d_{opt,rnd})$, optimal solutions share many edges with the MST.

Figures 1(a) and 1(b) summarize the results from Tables 2, 3, and 4. They plot $d_{opt,rnd}$ ($d_{opt,rnd}/n$) and $d_{mst,opt}$ ($d_{mst,opt}/n$) using either random w_{ij} or Euclidean w_{ij} over n. Furthermore, Figure 2 shows the distribution of $d_{opt,rnd}$ and $d_{mst,opt}$ (random and Euclidean w_{ij}) for 1000 randomly

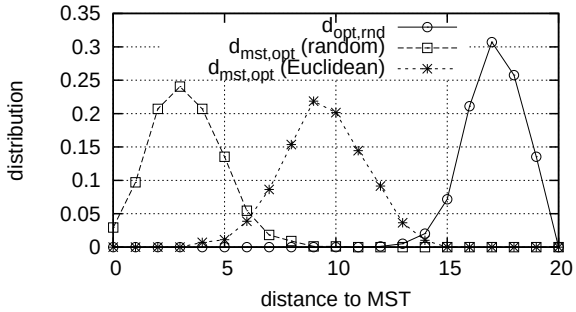


Figure 2 We show for 1000 randomly generated instances of 20-node OCST problems the distribution of $d_{opt,rnd}$ and $d_{mst,opt}$ for random and Euclidean w_{ij}.

generated instances of 20-node OCST problems. Optimal solutions and MSTs share a larger number of edges than optimal solutions and random solutions.

4. Designing Heuristic Optimization Methods for OCST Problems

The similarity between optimal solutions for OCST problems and MSTs can be of importance for the design of optimization methods as it is well known that their performance can be increased if problem-specific knowledge is considered. Therefore, we can make use of this observation and design efficient and problem-specific heuristic optimization methods for the OCST problem. Important determinants of heuristic optimization methods are the initial solution (starting point), the problem representation, the search operators, and the evaluation of solutions. Consequently, four promising approaches on how to consider the similarity between optimal solutions and MSTs can be identified:

- Start with solutions that are similar to MSTs.
- Design redundant representations such that solutions similar to MSTs are overrepresented.
- Design search operators that favor trees similar to MSTs.
- Assign a higher fitness value to solutions similar to MSTs.

The following paragraphs discuss these four possibilities. The performance of many heuristic optimization approaches, especially local search methods, rely on the starting point of the search. Therefore, as $\mu(d_{mst,opt})$ is low it is a reasonable approach to start the search with solutions that are an MST or similar to it. We make use of this observation in Section 5 and examine how the performance of different heuristic search methods depend on the starting solution. The experimental results confirm that some heuristic optimization methods need either less fitness evaluations, or find better solutions when starting from an MST than starting from a random tree.

Using an MST as the starting solution has already been used for the design of efficient heuristics for other tree and tree-related problems. For example, for the metric traveling salesman problem (TSP), creating an Eulerian graph from an MST (each edge is taken twice) and producing an Eulerian tour on this graph results in a tour whose cost is at most twice the cost of the optimal tour. The 3/2-approximation algorithm from Christofides [11] follows this concept. It starts with an MST, computes a minimum cost perfect matching on the set of odd-degree vertices, and then generates an Eulerian graph. Also for the Steiner tree problem, using an MST as the starting solution is a promising strategy for heuristics as constructing an MST on the set of required nodes already results in a 2-approximation [10]. Therefore, different heuristics (e.g. the ER heuristic from Borah et al. [6]) start with an MST. Robins and Zelikovsky [40] presented an efficient polynomial-time approximation scheme for the Steiner tree problem that starts with an MST and results in the best-known performance ratio approaching $1 + \ln 3/2 \approx 1.55$. Hwang [20] showed for the rectilinear Steiner tree problem that an MST is a 3/2-approximation. Consequently, a variety of approaches either start with an MST and improve this solution in subsequent steps [18, 8, 21, 40], or imitate

the MST construction of Kruskal and Prim [2, 39]. For more information on heuristics for Steiner tree problems, compare Kahng and Robins [22] and Robins and Zelikovsky [40]. Another example where using an MST as the starting solution results in efficient heuristics is the capacitated MST problem. The objective is to find a minimum cost tree such that some capacity constraints (usually on node weights) are observed. As an MST is already the optimal solution if it is a feasible solution (capacity constraints are met), many simple heuristics consider edge weights and either construct a solution similar to an MST (e.g. the unified algorithm from Kershenbaum and Chou [24], or the modified Kruskal algorithm from Boorstyn and Frank [5]), or modify an infeasible MST solution (e.g. the start procedure of Elias and Ferguson [12]). Examples for heuristic optimization methods for the CMST problem that start with an MST are presented in Kershenbaum et al. [23], or Gavish et al. [15]. For a survey on heuristics for the CMST problem compare Amberg et al. [1] and Vo [46].

When using search methods or search heuristics, the solutions (trees) must be encoded using a representation. For redundant representations, the number of solutions in the search space exceeds the number of possible trees. Redundant representations are commonly used for heuristics like EAs and increase EA performance if solutions that are similar to optimal solutions are overrepresented [42]. Therefore, overrepresenting solutions that are similar to MSTs can increase the performance of heuristic search approaches for OCST problems. This principle was, for example, used for the link-and-node biased representation proposed by Palmer [30]. This is a redundant representation that overrepresents solutions similar to MSTs [14]. Consequently, when using this representation for OCST problems, EA performance can be increased [44].

Another possibility to increase search performance is to use search operators that favor MST-like trees. If this principle is applied to search approaches for the OCST problem, search operators do not undirectedly explore the search space, but prefer edges of low weight. This concept was implicitly used by Raidl and Julstrom [36] who proposed the edge-set representation, a direct representation of trees for the degree-constrained MST problem and other tree problems [45]. The search operators are combined with additional heuristics that prefer low-weight edges. As a result, the search operators favor solutions similar to MSTs which results in higher performance in comparison to search approaches that do not consider edge weights [36].

The final possibility is to modify the fitness evaluation of trees such that solutions that are similar to MSTs get an additional bonus. As a result, heuristic optimization methods would converge faster to MST-like solutions (if the quality of the solutions guides the search), and the performance of the search can be increased. Currently, the author is not aware of any approaches that apply this heuristic to the OCST problem.

Heuristic optimization algorithms that favor MST-like solutions can only be reasonable for problems where optimal solutions are similar to MSTs. This paper presents experimental evidence that OCST problems have this specific property. However, if there is no such problem-specific knowledge (e.g. all distance weights $w_{ij} = 1$ and all trees are MSTs), it makes no sense to use the proposed techniques. Therefore, methods that can solve OCST problems efficiently are not necessarily appropriate for other tree problems where optimal solutions are not similar to MSTs.

5. Performance of Heuristic Optimization Methods

We investigate how the performance of different heuristic optimization methods depend on the type of starting solution T_s. We show results for a greedy search strategy, a simulated annealing (SA) approach, local search (LS), and genetic algorithms (GA).

The greedy search starts with the initial solution T_s and iteratively examines all neighboring trees that are different in one edge and chooses the tree T_j with lowest cost $w(T_j)$. The neighborhood of T_i consists of all T_j that can be created by the change of one edge ($d_{ij} = 1$). The heuristic stops if

the current solution can not be improved any more. The number of different neighbors for a tree depends on the structure of the tree and varies between $(n-1)(n-2)$ and $\frac{1}{6}n(n-1)(n+1)-n+1$.

In the SA approach, a new, neighboring solution T_n replaces the original solution T_o if it has lower cost ($w(T_n) < w(T_o)$). Otherwise, T_n replaces T_o with probability $P(U) = \exp(\frac{w(T_o)-w(T_n)}{U})$. P depends on the temperature U which is reduced during each SA run according to a fixed cooling schedule. In each iteration, U is reduced by 0.995 ($U_i = 0.995 \times U_{i-1}$). With lowering U, the probability of accepting worse solutions decreases. The initial temperature U_0 is set with respect to the different test problems that should be solved. For each test problem, 1,000 random solutions T_i are generated before the SA run and the initial temperature is set to $U_0 = 0.1 \times \sigma(w(T_i))$, where $\sigma(w(T_i))$ denotes the standard deviation of the cost of the 1,000 randomly generated solutions T_i. U_0 is set relatively low as large values of U_0 would result in a high probability of accepting worse solutions at the beginning of an SA run and thus the performance of the search would become independent of T_s. Each SA run is stopped after $eval_m$ search steps.

In the LS approach, a new, neighboring solution replaces the original solution if it has lower cost. This approach is equivalent to a SA approach with $U = 0$. As for the other heuristics, LS is stopped after $eval_m$ search steps.

Raidl and Julstrom [36] presented two GA variants for solving tree problems that can be applied to OCST problems. Both variants are steady-state GAs with a population *pop* of size N. In each search step, a new solution T_{off} is created by recombination and mutation. Mutation is applied after crossover with probability p_{mut} to each edge of T_{off} and replaces it by a random edge. The resulting offspring T_{off} is inserted into the population *pop* if it has lower cost than the worst solution in the population ($w(T_{off}) < \max_{T \in pop} w(T)$). The two variants differ in the used initialization and crossover operators. In the first variant (denoted as GA) the initial population is created randomly and crossover creates a new offspring T_{off} from two randomly chosen parents $T_1, T_2 \in pop$ such that in a first step all edges $(E_1 \cap E_2)$ of both parents are included in T_{off}. Then, T_{off} is completed by randomly inserting edges from $(E_1 \cup E_2) \backslash (E_1 \cap E_2)$. Edges that would result in a cycle are discarded. The second variant (denoted as GA-ESH) uses heuristic initialization and crossover operators that favor low-weight edges. As recommended by Raidl and Julstrom [36], we use the heuristic initialization operator with $\alpha = 1.5$. With lower parameter α $(0 \leq \alpha \leq (n-1)/2)$, low-weight edges are encoded with higher probability [45]. The proposed heuristic crossover operator transfers all edges $(E_1 \cap E_2)$ that exist in both parents T_1 and T_2 to T_{off}. Then, the remaining edges are chosen randomly from $(E_1 \cup E_2) \backslash (E_1 \cap E_2)$ using a tournament with replacement of size two. This means, the distance weights of two edges are compared and the edge with the lower weight is inserted into T_{off} (if no cycle is created). GA-ESH represents the current state-of-the art approach for solving the OCST problem. For the experiments, we use a population size $N = 50$, a mutation probability of $p_{mut} = 1/n$, and stop the GA after $eval_m$ tree evaluations.

Peleg and Reshef [32] and Reshef [38] presented an $O(\log n)$ approximation for OCST problems with Euclidean distance weights and an $O(\log^3 n)$ approximation for arbitrary (non-Euclidean) distance weights. They developed a recursive construction algorithm which uses a partitioning algorithm that clusters the nodes in different partitions. The proposed algorithms are the best approximation algorithms that are currently available for OCST problems.

In the experimental study we compare the performance of different heuristic optimization methods for three different starting solutions T_s: either the solution generated by the approximation algorithm from Peleg and Reshef [32] (denoted as PeRe), or a random tree (denoted as rnd), or an MST. Table 5 presents the results for the test instances from the literature (compare Section 2.2). It shows for different starting solutions T_s the cost $w(T)$ of 1,000 randomly generated solutions of type T_s (denoted as random), the performance (average cost of best found solution and average number of evaluations) of a greedy search starting with T_s, the maximum number $eval_m$ of evaluations for SA, LS, GA, and GA-ESH, and the average cost of the best found solution for SA, LS,

	T_s	random cost	greedy cost (%opt)	greedy *eval*	$eval_m$	SA cost (%opt)	LS cost (%opt)	GA cost (%opt)	GA-ESH cost (%opt)
palm6	MST	709,770	693,180 (0%)	57	500	698,644 (0.8%)	699,740 (0.9%)	693,680 (0.07%)	693,180 (0%)
	rnd	1,732,240	699,442 (0.9%)	186		700,150 (1.0%)	700,635 (1.1%)	693,781 (0.09%)	
	PeRe	926,872	695,566 (0.3%)	75		700,927 (1.1%)	697,951 (0.7%)	693,478 (0.04%)	
palm12	MST	3,876,488	3,541,915 (3.3%)	1,794	500	3,579,384 (4.4%)	3,498,458 (2.0%)	3,730,657 (8.8%)	3,623,253 (5.7%)
	rnd	11,793,000	3,492,357 (1.9%)	5,752		3,594,002 (4.8%)	3,512,364 (2.5%)	4,004,888 (17%)	
	PeRe	4,663,256	3,522,282 (2.7%)	2,806		3,580,330 (4.4%)	3,502,914 (2.2%)	3,912,014 (14%)	
palm24	MST	1,959,790	1,086,656 (0%)	57,654	2,500	1,098,379 (1.1%)	1,092,353 (0.5%)	1,154,449 (6.2%)	1,250,073 (15%)
	rnd	10,348,900	1,086,656 (0%)	92,549		1,097,179 (1.0%)	1,098,732 (1.1%)	1,226,783 (13%)	
	PeRe	2,322,022	1,086,656 (0%)	39,303		1,103,834 (1.6%)	1,096,615 (0.9%)	1,181,316 (8.7%)	
raidl10	MST	58,352	53,674 (0%)	435	500	54,762 (2.0%)	53,699 (0.05%)	57,141 (6.5%)	55,761 (3.9%)
	rnd	328,993	53,674 (0%)	2,296		54,663 (1.8%)	54,009 (0.6%)	67,893 (26%)	
	PeRe	194,097	53,674 (0%)	1,711		54,796 (2.1%)	53,674 (0%)	70,120 (31%)	
raidl20	MST	168,022	157,570 (0%)	3,530	2,500	158,983 (0.9%)	157,570 (0%)	159,911 (1.5%)	158,974 (0.9%)
	rnd	1,961,980	157,995 (0.3%)	50,843		161,023 (2.2%)	160,214 (1.7%)	205,718 (30.6%)	
	PeRe	849,796	158,704 (0.7%)	32,053		160,943 (2.1%)	160,578 (1.9%)	209,731 (33%)	
raidl50	MST	912,303	809,311 (0.30%)	211,394	10,000	829,780 (2.84%)	811,098 (0.52%)	852,091 (5.6%)	880,927 (9.18%)
	rnd	20,925,600	806,946 (0.01%)	1,796,006		864,736 (7.17%)	887,066 (9.94%)	1,541,047 (91%)	
	PeRe	5,913,850	807,353 (0.06%)	1,052,945		890,082 (10.3%)	883,483 (9.50%)	1,488,774 (85%)	
raidl75	MST	2,403,577	1,717,491 (0%)	1,369,331	10,000	2,042,603 (19%)	1,852,905 (7.9%)	1,971,638 (15%)	2,003,433 (17%)
	rnd	58,821,400	1,717,491 (0%)	8,445,914		2,401,151 (40%)	2,330,226 (36%)	8,814,074 (413%)	
	PeRe	13,396,495	1,749,322 (1.9%)	4,573,544		2,370,276 (38%)	2,303,405 (34%)	4,957,022 (189%)	
raidl100	MST	3,607,091	2,561,543 (0%)	2,454,388	40,000	2,713,040 (6%)	2,619,256 (2.3%)	2,831,167 (11%)	2,935,381 (14.6%)
	rnd	118,702,000	2,603,146 (1.6%)	22,730,365		2,870,197 (12%)	2,937,911 (15%)	5,200,334 (103%)	
	PeRe	24,429,677	2,709,603 (5.8%)	12,154,877		2,941,064 (15%)	2,959,953 (16%)	4,620,145 (80%)	
berry6	MST	534	534 (0%)	28	500	534 (0%)	534 (0%)	534 (0%)	534 (0%)
	rnd	1,284	534 (0%)	207		534 (0%)	534 (0%)	534 (0.07%)	
	PeRe	842	534 (0%)	120		534 (0%)	534 (0%)	536 (0.3%)	
berry35	MST	16,915	16,915 (0%)	3,387	2,500	21,818 (29%)	16,915 (0%)	16,915 (0%)	16,915 (0%)
	rnd	379,469	16,915 (0%)	467,008		22,426 (33%)	22,642 (34%)	44,661 (164%)	
	PeRe	60,382	16,915 (0%)	171,425		21,563 (27%)	20,777 (23%)	31,765 (88%)	

Table 5 We compare the performance of different heuristic optimization methods using different starting solutions T_s for test instances from the literature. For greedy search, on average the cost of the best found solution is similar, however, the number of evaluations are lowest when starting from an MST. For SA, LS, and GA, starting from an MST on average results in better solutions than starting from a random tree.

GA, and GA-ESH. For the population-based GA, one randomly chosen individual in the initial population of size N is set to T_s. Therefore, this randomly chosen solution is either an MST, a randomly chosen tree, or the result of the PeRe-approximation. All other solutions in the initial population are random trees. 50 independent runs are performed for each problem instance and the gap $(w(T^{bf}) - w(T^{best}))/w(T^{best})$ (in %) between the best found solution T^{bf} and the optimal solution T^{best} (compare Table 1) is shown in brackets.

For all problem instances the average cost of randomly created solutions and PeRe is greater than the cost of an MST. For the greedy heuristic there are no great differences between the cost of the best found solution for different T_s. However, there are large differences in the average number *eval* of trees that are evaluated during a run of the greedy search. Starting from a randomly generated tree always results in the highest *eval*; starting from an MST always results (except for palmer24) in the lowest *eval*. On average, *eval* for starting from PeRe is between the results for starting from an MST and starting from a random tree. For SA, LS, and GA, starting with an MST always results (except for palmer24 and raidl10) in better solutions than starting from a random tree, and often in better solutions than starting from PeRe. Using SA or LS and starting from an MST always results (except palmer6 and berry35) in the same (berry6) or better performance than GA-ESH. Summarizing the results, the cost of an MST is on average lower than the cost of the PeRe-approximation (up to a factor of 6 for raidl100) and the cost of a random tree (up to a factor of 35 for raidl100). The different heuristic optimization methods find better solutions (up to a factor of 4 for raidl75) when starting the search from an MST in comparison to starting from a random tree.

r_{ij}	n	T_s	random cost	Greedy cost (%opt)	Greedy eval	SA cost (%opt)	LS cost (%opt)	GA cost (%opt)	GA-ESH cost (%opt)
uniform]0,10]	10	MST	1,670	1,515 (0%)	698	1,527 (0.80%)	1,539 (1.58%)	1,605 (5.94%)	1,574 (3.89%)
		rnd	3,284	1,520 (0.32%)	2,527	1,555 (2.65%)	1,550 (2.28%)	1,673 (10.4%)	
		PeRe	2,114	1,519 (0.23%)	1,475	1,530 (1.00%)	1,549 (2.20%)	1,665 (9.89%)	
	25	MST	10,794	8,839 (0%)	41,854	8,937 (1.11%)	8,885 (0.52%)	9,230 (4.43%)	9,412 (6.49%)
		rnd	40,261	8,844 (0.06%)	122,463	9,004 (1.88%)	9,027 (2.14%)	10,337 (17.0%)	
		PeRe	12,827	8,855 (0.18%)	62,073	9,021 (2.06%)	9,044 (2.33%)	11,088 (25.5%)	
	50	MST	61,450	44,142 (0%)	786,904	44,749 (1.38%)	44,600 (1.04%)	45,320 (2.67%)	47,391 (7.36%)
		rnd	207,139	73,179 (65.8%)	999,800	45,187 (2.37%)	45,364 (2.77%)	50,336 (14.0%)	
		PeRe	63,638	45,301 (2.63%)	778,602	45,248 (2.51%)	45,312 (2.65%)	51,592 (16.9%)	
	100	MST	272,987	256,149 (42%)	10^7	181,538 (0.5%)	180,592 (0%)	182,787 (1.2%)	189,559 (4.97%)
		rnd	1,207,970	1,069,412 (492%)	10^7	183,188 (1.4%)	182,706 (1.2%)	194,423 (7.7%)	
		PeRe	251,198	255,158 (41%)	10^7	182,419 (1.0%)	183,280 (1.5%)	197,270 (9.2%)	
	150	MST	645,042	624,139 (56%)	10^7	399,312 (0.0%)	399,262 (0%)	401,127 (0.5%)	417,045 (4.45%)
		rnd	3,390,500	3,082,983 (672%)	10^7	403,119 (1.0%)	402,687 (0.9%)	425,814 (6.7%)	
		PeRe	562,282	563,160 (41%)	10^7	401,615 (0.6%)	401,946 (0.7%)	434,682 (8.9%)	
Zipf [1,...,10]	10	MST	1,082	985 (0%)	670	994 (0.86%)	1,004 (1.89%)	1,047 (6.24%)	1,027 (4.21%)
		rnd	1,910	986 (0.12%)	2,537	1,014 (2.87%)	1,009 (2.45%)	1,090 (10.6%)	
		PeRe	1,414	986 (0.10%)	1,511	996 (1.06%)	1,009 (2.45%)	1,085 (10.1%)	
	25	MST	8,771	7,164 (0%)	42,900	7,262 (1.37%)	7,224 (0.84%)	7,484 (4.47%)	7,637 (6.61%)
		rnd	28,194	7,176 (0.17%)	122,733	7,323 (2.22%)	7,328 (2.30%)	8,375 (16.9%)	
		PeRe	10,398	7,182 (0.26%)	61,930	7,322 (2.20%)	7,337 (2.42%)	8,998 (25.6%)	
	50	MST	41,650	30,253 (0%)	749,451	30,411 (0.52%)	30,380 (0.42%)	30,881 (2.08%)	32,315 (6.82%)
		rnd	142,800	49,664 (64.16%)	10^7	30,672 (1.39%)	30,721 (1.55%)	34,007 (12.4%)	
		PeRe	42,980	31,001 (2.47%)	781,995	30,606 (1.17%)	30,738 (1.60%)	35,536 (17.5%)	
	100	MST	197,763	180,861 (43%)	10^7	127,529 (0.5%)	126,865 (0%)	128,139 (1.0%)	133,760 (5.44%)
		rnd	830,401	739,332 (482%)	10^7	128,215 (1.1%)	127,725 (0.7%)	137,119 (8.1%)	
		PeRe	176,252	180,780 (43%)	10^7	128,681 (1.4%)	127,274 (0.3%)	140,538 (11%)	
	150	MST	447,495	431,925 (56%)	10^7	277,124 (0%)	277,474 (0.1%)	281,240 (1.5%)	290,765 (4.92%)
		rnd	2,311,510	2,131,665 (669%)	10^7	280,049 (1.1%)	277,943 (0.3%)	292,757 (5.6%)	
		PeRe	425,312	420,659 (52%)	10^7	279,536 (0.9%)	280,652 (1.3%)	299,241 (8.0%)	

Table 6 We compare the performance of different heuristic optimization methods using different starting solutions T_s for randomly created OCST test instances with Euclidean distance weights. When starting from an MST, greedy search needs either less evaluations ($n < 100$) or finds better solutions ($n \geq 100$) than starting from a random solution. SA, LS, and GA starting from an MST find better solutions than starting from a random tree or from PeRe.

Tables 6 and 7 extend the analysis and show results for randomly generated OCST problem instances with 10, 25, 50, 100, and 150 nodes. As before, we compare the average cost of an MST, 1,000 randomly generated trees, and the PeRe-approximation (denoted as random). Furthermore, we present results (average cost of best found solution and average number of evaluations) for a greedy search using different types of T_s and for SA, LS, GA, and GA-ESH the average cost of the best solution found after $eval_m$ evaluations (compare Table 8). The values in brackets show the gap (in %) between the average cost of the best found solution and the overall best found solution. For each size and type, we generated 100 random problem instances (for $n \geq 100$ we only generated 50 problem instances) and performed 20 runs of the different heuristics for different T_s. Due to computational restrictions we limited the number of search steps of the greedy search to 1,000,000.

For the results in Table 6 we generated random problem instances with Euclidean distance weights. The nodes are placed randomly on a two-dimensional grid of size 10x10. For the results in Table 7 the distance weights are randomly generated and uniformly distributed in]0,10]. For both tables, the demands r_{ij} are randomly generated and either uniformly distributed in]0,10] or Zipf distributed with $z = 1$ and $N = 10$. The Zipf distribution [47] is a power law probability distribution and describes natural phenomenas like Internet traffic, population distribution, or

r_{ij}	n	T_s	random cost	Greedy cost (%opt)	Greedy eval	SA cost (%opt)	LS cost (%opt)	GA cost (%opt)	GA-ESH cost (%opt)
uniform]0, 10]	10	MST	717	676 (0%)	291	686 (1.37%)	680 (0.49%)	703 (3.93%)	689 (1.95%)
		rnd	2,863	677 (0.10%)	2,472	691 (2.11%)	685 (1.25%)	852 (25.9%)	
		PeRe	1,949	677 (0.11%)	1,661	687 (1.62%)	684 (1.10%)	855 (26.4%)	
	25	MST	3,094	2,720 (0%)	12,420	2,806 (3.16%)	2,738 (0.65%)	2,840 (4.40%)	2,855 (4.93%)
		rnd	37,254	2,723 (0.08%)	126,859	2,893 (6.34%)	2,882 (5.93%)	4,894 (79.9%)	
		PeRe	14,000	2,723 (0.09%)	74,831	2,885 (6.03%)	2,864 (5.29%)	5,290 (94.5%)	
	50	MST	7,921	6,663 (0%)	169,398	6,989 (4.89%)	6,753 (1.36%)	7,081 (6.28%)	7,205 (8.14%)
		rnd	214,238	45,822 (587%)	10^7	7,447 (11.8%)	7,435 (11.6%)	16,009 (140%)	
		PeRe	59,304	8,968 (34.6%)	949,768	7,492 (12.4%)	7,405 (11.1%)	14,523 (118%)	
	100	MST	21,719	18,436 (9.22%)	10^7	17,584 (4.2%)	16,879 (0%)	18,020 (6.8%)	18,561 (10.0%)
		rnd	1,206,920	990,391 (5,767%)	10^7	18,882 (11%)	19,765 (17%)	42,324 (151%)	
		PeRe	244,349	187,958 (1,014%)	10^7	19,003 (13%)	19,432 (15%)	36,938 (119%)	
	150	MST	40,596	39,184 (29%)	10^7	31,451 (4.0%)	30,247 (0%)	32,807 (8.5%)	33,350 (10.3%)
		rnd	3,290,630	3,094,862 (10,131%)	10^7	36,107 (19%)	36,712 (21%)	84,711 (180%)	
		PeRe	549,068	505,367 (1,570%)	10^7	37,417 (24%)	37,355 (24%)	71,298 (136%)	
Zipf $[1, \ldots, 10]$	10	MST	486	460 (0.04%)	306	465 (1.32%)	459 (0%)	474 (3.11%)	464 (1.10%)
		rnd	1,998	460 (0.06%)	2,519	466 (1.50%)	463 (0.75%)	580 (26.27%)	
		PeRe	1,285	460 (0.08%)	1,643	467 (1.67%)	462 (0.54%)	577 (25.6%)	
	25	MST	2,191	1,930 (0%)	12,643	1,995 (3.38%)	1,945 (0.79%)	2,031 (5.23%)	2,045 (5.96%)
		rnd	26,128	1,932 (0.12%)	124,427	2,054 (6.40%)	2,045 (5.97%)	3,475 (80.0%)	
		PeRe	9,664	1,931 (0.03%)	73,930	2,045 (5.98%)	2,041 (5.74%)	3,702 (91.8%)	
	50	MST	5,389	4,422 (0%)	182,112	4,665 (5.49%)	4,486 (1.45%)	4,742 (7.26%)	4,827 (9.17%)
		rnd	147,659	31,930 (622%)	10^7	4,999 (13.1%)	4,968 (12.4%)	10,704 (142%)	
		PeRe	39,929	6,084 (37.6%)	951,344	4,992 (12.9%)	4,953 (12.0%)	9,936 (125%)	
	100	MST	14,528	12,631 (12%)	10^7	11,735 (4.0%)	11,280 (0%)	11,997 (6.4%)	12,230 (8.4%)
		rnd	829,853	693,941 (6,052%)	10^7	13,375 (19%)	13,237 (17%)	29,777 (164%)	
		PeRe	167,363	129,809 (1,050%)	10^7	12,984 (15%)	12,997 (15%)	25,415 (125%)	
	150	MST	24,995	23,796 (25%)	10^7	19,926 (5.1%)	18,964 (0%)	20,540 (8.3%)	20,817 (9.77%)
		rnd	2,243,520	2,213,067 (11,569%)	10^7	22,847 (20%)	22,698 (20%)	52,773 (178%)	
		PeRe	375,782	350,425 (1,747%)	10^7	23,279 (22%)	23,490 (24%)	47,312 (149%)	

Table 7 We compare the performance of different heuristic optimization methods using different starting solutions T_s for randomly created OCST test instances with distance weights uniformly distributed in $]0, 10]$. When starting from an MST, greedy search either needs less evaluations ($n < 100$) or finds better solutions ($n \geq 100$) than starting from a random solution or from PeRe. SA, LS, and GA starting from an MST find better solutions than starting from a random tree or from PeRe.

n	10	25	50	100	150
$eval_m$	500	2,500	10,000	40,000	80,000

Table 8 Maximum number of search steps for different problem sizes

word usage [26, 34]. A discrete Zipf distribution is defined by $P_z(x)$, which denotes the probability of $x \in \{1, 2, \ldots, N\}$:

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

The results for the randomly created OCST problem instances are similar to the results for the problem instances from the literature. The average cost of an MST is always lower than the average cost of random solutions (significant using a t-test with an error level of 0.01). Because for Euclidean distance weights (Table 6) the PeRe approximation is more tight and because optimal solutions have a larger average distance $\mu(d_{mst,opt})$ towards an MST (compare Figure 1), the PeRe approximation results in better solutions and the results are similar to an MST. When using random distance

weights (Table 7), the average cost of an MST is always lower than the average cost of the PeRe approximation (also significant using a t-test with an error level of 0.01).

For Euclidean distance weights (Table 6), greedy search starting from an MST either finds good solutions faster ($n < 100$), or finds better solutions ($n \geq 100$) than starting from a random tree. With increasing n, greedy search starting from PeRe performs similar to when starting from an MST. For SA, LS, and GA, starting from an MST always finds better solutions than starting from a random solution or from PeRe. Furthermore, starting from an MST results in better results than GA-ESH (except $n = 10$).

For uniformly distributed distance weights (Table 7) starting from an MST always results in better performance than starting from a random tree or PeRe (significant using a t-test with an error level of 0.01, except SA with $n = 10$). Furthermore, using SA, LS, or GA and starting with an MST results in better performance than GA-ESH (except $n = 10$).

6. Conclusions

By analyzing the properties of optimal solutions, we have experimentally found for test instances from the literature, and random OCST problems with either uniform or Euclidean distance weights, that the distances between optimal solutions and MSTs are on average lower than the distances between optimal solutions and randomly created trees. Furthermore, optimal solutions for problems with random distance weights are more similar to MSTs than optimal solutions for problems with Euclidean distance weights. Consequently, the performance of several heuristic optimization methods can be increased if solutions similar to MSTs are preferred. Doing this, either the initial solution, the problem representation, the search operators, or the evaluation of solutions can be modified such that solutions similar to MSTs are favored. Results presented for a simple greedy heuristic using different starting solutions show that starting from MSTs results in a lower number of search steps in comparison to starting from random trees or starting from solutions generated by an approximation algorithm. Starting a local search, a simulated annealing approach, and a genetic algorithm from MSTs also increases solution quality in comparison to starting either from a random solution or from a solution generated by an approximation algorithm. Starting from an MST also finds better solutions than existing state-of-the-art heuristics.

Acknowledgments
The author thanks the referees and the associate editor for valuable comments on the paper. A major part of this work was done while the author was with the University of Mannheim.

References

[1] Amberg, A., W. Domschke, Vo. 1996. Capacitated minimum spanning trees: Algorithms using intelligent search. *Comb. Opt.: Theory and Practice* **1** 9–39.

[2] Bern, M. W. 1988. Two probabilistic results on rectilinear steiner trees. *Algorithmica* **3**(2) 191–204.

[3] Berry, L. T. M., B. A. Murtagh, G. McMahon. 1995. Applications of a genetic-based algorithm for optimal design of tree-structured communication networks. *Proceedings of the Regional Teletraffic Engineering Conference of the International Teletraffic Congress*. Telkom South Africa, Pretoria, South Africa, 361–370.

[4] Berry, L. T. M., B. A. Murtagh, S. J. Sugden. 1994. A genetic-based approach to tree network synthesis with cost constraints. Hans Jürgen Zimmermann, ed., *Second European Congress on Intelligent Techniques and Soft Computing - EUFIT'94*, vol. 2. Verlag der Augustinus Buchhandlung, Promenade 9, D-52076 Aachen, 626–629.

[5] Boorstyn, R. R., H. Frank. 1977. Large-scale network topological optimization. *IEEE Transactions on Communications* **25** 29–47.

[6] Borah, M., R. M. Owens, M. J. Irwin. 1999. A fast and simple steiner routing heuristic. *Discrete Applied Mathematics* **90** 51–67.

[7] Cayley, A. 1889. A theorem on trees. *Quarterly Journal of Mathematics* **23** 376–378.

[8] Chao, Ting-Hai, Yu-Chin Hsu. 1994. Rectlinear steiner tree construction by local and global refinement. *IEEE Transactions on Computer-Aided Design of Integrated Circutits and Systems* **13**(3) 303–309.

[9] Chou, H., G. Premkumar, Chao-Hsien Chu. 2001. Genetic algorithms for communications network design - an empirical study of the factors that influence performance. *IEEE Transactions on Evolutionary Computation* **5**(3) 236–249.

[10] Choukhmane, E.-A. 1978. Une heuristique pour le problème de l'arbre de steiner. *RAIRO Rech. Opér.* **12** 207–212.

[11] Christofides, N. 1976. Worst-case analysis of a new heuristic for the traveling salesman problem. Tech. Rep. 388, Graduate School of Industratial Administration, Carnegie-Mellon University, Pittsburgh, PA.

[12] Elias, D., M. J. Ferguson. 1974. Topological design of multipoint teleprocessing networks. *IEEE Transactions on Communications* **22** 1753–1762.

[13] Garey, Michael R., David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

[14] Gaube, Thomas, Franz Rothlauf. 2001. The link and node biased encoding revisited: Bias and adjustment of parameters. Egbert J. W. Boers, Stefano Cagnoni, Jens Gottlieb, Emma Hart, Pier Luca Lanzi, Günther R. Raidl, Robert E. Smith, Harald Tijink, eds., *Applications of Eolutionary Computing: Proc. EvoWorkshops 2001*. Springer, Berlin, 1–10.

[15] Gavish, B., Chung-Lun Li, D. Simchi-Levi. 1992. Analysis of heuristics for the design of tree networks. *Annals of Operations Research* **36** 77–86.

[16] Goldberg, D. E. 1989. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA.

[17] Harik, G. R., E. Cantú-Paz, D. E. Goldberg, B. L. Miller. 1997. The gambler's ruin problem, genetic algorithms, and the sizing of populations. T. Bäck, ed., *Proceedings of the Forth International Conference on Evolutionary Computation*. IEEE Press, New York, 7–12.

[18] Ho, J.-M., G. Vijayan, C. K. Wong. 1990. New algorithms for the rectilinear steiner tree problem. *IEEE Transactions on Computer-Aided Design* **9** 185–193.

[19] Hu, T. C. 1974. Optimum communication spanning trees. *SIAM Journal on Computing* **3**(3) 188–195.

[20] Hwang, F. K. 1976. On steiner minimal trees with rectilinear distance. *SIAM J. Applied Math.* **1** 104–114.

[21] Julstrom, B. 2002. A scalable genetic algorithm for the rectilinear steiner problem. *Proceedings of the 2002 Congress on Evolutionary Computation CEC'02*. IEEE Press, Piscataway, NJ, 1169–1173.

[22] Kahng, A. B., G. Robins. 1992. On performance bounds for a class of rectilinear Steiner tree heuristics in arbitrary dimension. *IEEE Transactions on Computer-Aided Design* **11**(11) 1462–1465.

[23] Kershenbaum, A., R. R. Boorstyn, R. Oppenheim. 1980. Second-order greedy algorithms for centralized teleprocessing network design. *IEEE Transactions on Communications* **28** 1835–1838.

[24] Kershenbaum, A., W. Chou. 1974. A unified algorithm for designing multidrop teleprocessing networks. *IEEE Transactions on Communications* **22** 1762–1772.

[25] Kim, J. R., M. Gen. 1999. Genetic algorithm for solving bicriteria network topology design problem. Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, Ali Zalzala, William Porto, eds., *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*. IEEE Press, 2272–2279.

[26] Li, W. 1992. Random texts exhibit zipf's-law-like word frequency distribution. *IEEE Transactions on Information Theory* **38**(6) 1843–1845.

[27] Li, Y. 2001. An effective implementation of a direct spanning tree representation in GAs. Egbert J. W. Boers, Stefano Cagnoni, Jens Gottlieb, Emma Hart, Pier Luca Lanzi, Günther R. Raidl, Robert E. Smith, Harald Tijink, eds., *Applications of evolutionary Computing: Proc. EvoWorkshops 2001*. Springer, Berlin, 11–19.

[28] Li, Y., Y. Bouchebaba. 1999. A new genetic algorithm for the optimal communication spanning tree problem. C. Fonlupt, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, eds., *Proceedings of Artificial Evolution: Fifth European Conference*. Springer, Berlin, 162–173.

[29] Neville, E. H. 1953. The codifying of tree-structure. *Proceedings of the Cambridge Philosophical Society* **49** 381–385.

[30] Palmer, C. C. 1994. An approach to a problem in network design using genetic algorithms. unpublished PhD thesis, Polytechnic University, Troy, NY.

[31] Papadimitriou, C. H., M. Yannakakis. 1991. Optimization, approximation, and complexity classes. *J. Comput. System Sci.* **43** 425–440.

[32] Peleg, David, Eilon Reshef. 1998. Deterministic polylog approximation for minimum communication spanning trees. *Lecture Notes in Computer Science* **1443** 670–682.

[33] Picciotto, S. 1999. How to encode a tree. Ph.D. thesis, University of California, San Diego, USA.

[34] Poosala, V. 1995. Zipf's law. Tech. rep., University of Wisconsin, Madison.

[35] Prüfer, H. 1918. Neuer Beweis eines Satzes über Permutationen. *Archiv für Mathematik und Physik* **27** 742–744.

[36] Raidl, G. R., B. A. Julstrom. 2003. Edge-sets: An effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation* **7**(3) 225–239.

[37] Raidl, Günther R. 2001. Various instances of optimal communication spanning tree problems. personal communciation.

[38] Reshef, E. 1999. Approximating minimum communication cost spanning trees and related problems. Master's thesis, Feinberg Graduate School of the Weizmann Institute of Science, Rehovot 76100, Israel.

[39] Richards, D. 1989. Fast heuristic algorithms for rectilinear steiner trees. *Algorithmica* **4** 191–207.

[40] Robins, Gabriel, Alexander Zelikovsky. 2005. Tighter bounds for graph steiner tree approximation. *SIAM J. Discret. Math.* **19**(1) 122–134. doi:http://dx.doi.org/10.1137/S0895480101393155.

[41] Rothlauf, F. 2006. *Representations for Genetic and Evolutionary Algorithms*. 2nd ed. Springer, Heidelberg.

[42] Rothlauf, F., D. E. Goldberg. 2003. Redundant representations in evolutionary computation. *Evolutionary Computation* **11**(4) 381–415.

[43] Rothlauf, F., D. E. Goldberg, A. Heinzl. 2002. Network random keys – A tree network representation scheme for genetic and evolutionary algorithms. *Evolutionary Computation* **10**(1) 75–97.

[44] Rothlauf, Franz, Armin Heinzl. 2004. Developing efficient metaheuristics for communication network problems by using problem-specific knowledge. Tech. Rep. 9/2004, Department of Information Systems, University Mannheim.

[45] Tzschoppe, C., F. Rothlauf, H.-J. Pesch. 2004. The edge-set encoding revisited: On the bias of a direct representation for trees. Deb, Kalyanmoy et al., ed., *Proceedings of the Genetic and Evolutionary Computation Conference 2004*. Springer, Heidelberg, 1174–1185.

[46] Vo, S. 2001. Capacitated minimum spanning trees. C. A. Floudas, P. M. Pardalos, eds., *Encyclopedia of Optimization*, vol. 1. Kluwer, Boston, 225–235.

[47] Zipf, G. K. 1949. *Human Behaviour and the Principle of Least-Effort*. Addison-Wesley, Cambridge, MA.