JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Edge Orientation and the Design of problem-specific EAs for the OCST problem

Wolfgang Steitz, Franz Rothlauf

Working Paper 04/2009
September 2009

Working Papers in Information Systems and Business Administration

Johannes Gutenberg-University Mainz
Department of Information Systems and Business Administration
D-55128 Mainz/Germany
Phone +49 6131 39 22734, Fax +49 6131 39 22185
E-Mail: sekretariat[at]wi.bwl.uni-mainz.de
Internet: `http://wi.bwl.uni-mainz.de`

Edge Orientation and the Design of problem-specific EAs for the OCST problem

Wolfgang Steitz and Franz Rothlauf, *Member, IEEE*

***Abstract*—We study the Euclidean variant of the optimal communication spanning tree (OCST) problem. An experimental analysis reveals that edges in optimal trees do not have only low distance weights but also point with higher probability towards the graph's center. We use this characteristic of optimal solutions for the design of problem-specific evolutionary algorithms (EAs). We extend recombination operators of direct encodings like edge-set and NetDir such that they prefer not only edges with low 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.**

***Index Terms*—optimal communications spanning tree problem, edge-set, NetDir, problem analysis, systematic design.**

I. Introduction

THE OPTIMAL COMMUNICATION spanning tree (OCST) problem [1] 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 solution approaches for the problem [2], [3]. Current state-of-the-art approaches for solving OCST problems are based on heuristics and metaheuristics, in particular evolutionary algorithms (EA).

The edge-set encoding [4] is a direct representation for trees which 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 non-heuristic. Both, mutation and crossover, are based on a randomized version of Kruskal's algorithm. Another direct representation for OCST problems is the NetDir encoding [2]. The encoding's crossover operator copies subtrees from parents to offspring. In the original specification, no heuristics are incorporated into the search operators.

High-quality solutions to Euclidean OCST problem instances contain disproportionately many edges that point towards the centers of their trees. We make use of this observation and systematically design problem-specific EAs. We develop efficient recombination operators for edge-set and NetDir encoding, such that not only distance weights but also edge orientation are used for constructing offspring trees. Experimental results reveal improved EA performance.

The main findings of this paper are:

1) Edges pointing toward the graphs' centers are overrepresented in good solutions to Euclidean OCST problem instances.
2) 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 graph's center.
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 OCST problems, present properties of optimal solutions, list various OCST test instances, and describe how to determine optimal solutions for small problem instances. In Section II-C, we study edge orientation in optimal solutions. Section III develops recombination operators that consider edge weights and orientation for edge-set and NetDir encodings. Section IV studies EA performance for different types of test instances.

II. The OCST Problem

A. Problem Definition

The OCST problem introduced by Hu [1] is a common combinatorial tree optimization problem. Given a distance and a demand matrix, it seeks a spanning tree which connects all given nodes and satisfies their communication requirements for minimum total cost.

Let $G = (V, E)$ be a 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

$$w(T) = \sum_{i,j \in V} r_{ij} pl_{ij}, \tag{1}$$

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 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 $w(T) \leq w(T')$ for all other spanning trees $w(T')$.

Manuscript received XXXX XX, 2009; revised XXXXX XX, 20XX.

W. Steitz is with Department of Information Systems and Business Administration, University of Mainz, Germany (e-mail: steitzw@uni-mainz.de)

F. Rothlauf is with Department of Information Systems and Business Administration, University of Mainz, Germany (e-mail: rothlauf@uni-mainz.de)

The OCST problem is $\mathcal{NP}$-hard [5, p. 207]. Furthermore, Reshef [6] showed that the problem is $\mathcal{MAX}$ $\mathcal{SNP}$-hard. Therefore, no polynomial-time approximation scheme exists, unless $\mathcal{NP} = \mathcal{P}$ [7]. Exact polynomial-time algorithms exist only for the restricted version of the problem [8]. Various approximation algorithms have been developed [9]–[11]; 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 [4], [12]–[16]. For an overview of EAs for the OCST problem, see Rothlauf [2].

To measure the difference between two spanning trees T_i and T_j, we count the number of edges that do not exist in both trees. Thus, the distance $d_{ij} \in \{0, 1, \ldots, n-1\}$ can be calculated as

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

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

B. Experimental Design

In our studies, we follow Raidl and Julstrom [4] and use randomly created OCST test instances. For our test instances, real-valued distance weights w_{ij} are Euclidean distances between nodes i and j which are randomly placed on a 2-dimensional grid of size 10×10.

We use two different types of demand. First, real-valued demands r_{ij} are randomly created and uniformly distributed in $]0, 10]$. Second, to create more realistic demand distributions, we create random demands following the Zipf distribution [17], a power law probability distribution. Many natural phenomena like Internet traffic, population distribution and word usage [18], [19] can be characterized by power law distributions. A discrete Zipf distribution is described by:

$$P_z(x) = \frac{1}{x^z} * \frac{1}{\sum_{i=1}^{N} \frac{1}{i^z}}, \tag{3}$$

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

For finding optimal, or at least near-optimal solutions of OCST problems, we use a mathematical programming solver for small problem instances with $n \leq 12$ and a GA for larger problem instances. The OCST problems are modeled as an integer linear program [20], 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. Therefore, we use an iterative GA for such problems. We choose the GA design in such a way that we can assume that the solution found 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. Then, T_0^{best} denotes the best solution that is found during the n_{iter} runs. In a second round, we again apply 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. Then, $n(T_i^{best})$ 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. For the encoding of trees, Network Random Keys (NetKeys) [14] are used. NetKeys have high locality [21], [22], which leads to high EA performance [2], 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 for finding optimal or near-optimal solutions is high.

We construct uniformly random trees via Prüfer numbers [23] using bijective mapping between Prüfer numbers and trees [24, pp. 103-104]. By generating random Prüfer numbers, this approach yields unbiased random spanning trees. In contrast, randomized versions of Kruskal's and Prim's algorithms would favor star-like trees [25].

C. Properties of Optimal Solutions

Optimal solutions for OCST problems are biased towards MSTs [26]. 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 [4], [27] make use of this fact by preferring edges with low distance weights [28].

We study below the orientation of edges in optimal solutions to identify additional properties of high-quality solutions. We find that edges directed towards a graph's center are overrepresented in optimal solutions. Such edges lead to shorter paths and hence to a lower cost solution.

How can we explain this observation? Kershenbaum [29] noticed that it is useful to run more traffic over nodes near the center of a tree than over nodes far away from the center. He distinguished between interior nodes (some traffic only transits) and leaf nodes (all traffic terminates). Consequently, edges near the tree's center carry much transit traffic whereas edges far away from the center carry almost no transit traffic. To construct a tree by favoring edges pointing towards the center 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 sense to consider only the orientation of edges that are some distance from the center of a tree since the orientation of edges connecting interior nodes is meaningless.

Figure 1 illustrates edge orientation. The orientation of an edge e_{ij} is the angle $\gamma \in [0, 90]$ between e_{ij} and the line

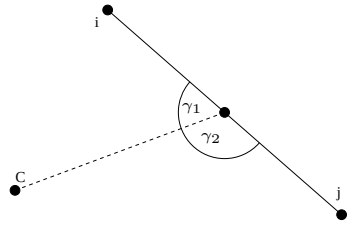
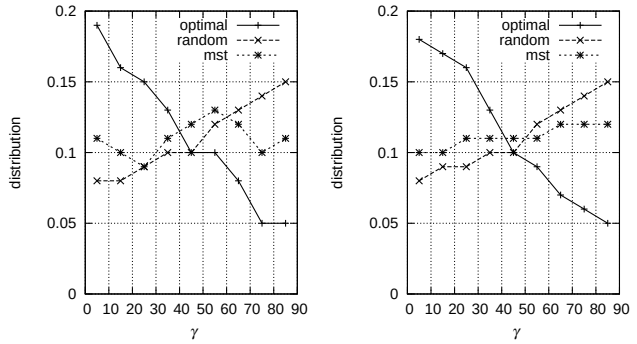


Fig. 1. Orientation of an edge e_{ij}



(a) $n = 15$, uniform demands (b) $n = 15$, Zipf demands

(c) $n = 20$, uniform demands (d) $n = 20$, Zipf demands

Fig. 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.

connecting the midpoint of e_{ij} and the center C of the tree. C is calculated as the average x-coordinates and y-coordinates of all nodes. Since $\gamma \leq 90$, the lower angle is chosen ($\gamma = min(\gamma_1, \gamma_2)$). For edges directly pointing to the center, $\gamma = 0$.

We compare edge orientation for optimal solutions, randomly created spanning trees, and MSTs. For each problem size, we create 1000 random Euclidean OCST test instances (compare Section II-B). For each OCST instance, we generate 10,000 random trees, calculate an MST, and determine an optimal (or near-optimal) solution as described in Section II-B.

Figure 2 presents results for $n = 15$ (Figs. 2a and 2b) and $n = 20$ (Figs. 2c and 2d). The demands are either uniformly distributed (Figs. 2a and 2c) or Zipf distributed (Figs. 2b and 2d). We plot the distribution of γ for optimal solutions ("optimal"), random solutions ("random"), and MSTs ("mst"). If the distribution is approximately uniform, orientation does not matter and all angles γ occur with 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 low γ occur more often. For $n = 20$, approximately 20% of all edges of optimal solutions have edge orientation $\gamma \leq 10$ whereas only approximately 4% of all edges have $\gamma > 80$. These results support the hypothesis that edges pointing towards the center of a tree are preferred in optimal solutions.

III. Problem-specific EAs for OCST Problems

Optimal solutions have a bias towards edges with low γ for both demand distributions, uniform and Zipf. We investigate whether heuristic optimization methods can make use of this observation by favoring edges with low distance weight *and* low orientation γ.

A. Direct Representations for Trees

1) Edge-set: The edge-set (ES) encoding [4], [27] is a direct representation which encodes trees as sets of edges. ES operators are either heuristic considering edge weights, or non-heuristic. Heuristic crossover operators result in higher performance in comparison to non-heuristic operators [4], [28].

Crossover creates an offspring from two parental trees $T_1 = (G, E_1)$ and $T_2 = (G, E_2)$ by iteratively selecting edges from $F = (E_1 \cup E_2)$. Therefore, offspring trees exist consisting solely of parental edges. Crossover operators differ according to the strategy used for selecting and inserting parental edges into offspring. Julstrom and Raidl [30] 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 [30], [31]. n-tournament edge-selection iteratively selects n edges, compares the associated edge weights, and inserts the edge with lowest weight into the offspring. This strategy has a bias towards low-weighted edges and MSTs [4], [28]. To strengthen the inheritance of common features from parents to offspring, edge-selection strategies can be designed as *-strategies [4]. 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.

2) NetDir: The NetDir encoding [2, Sect. 7.1] is a direct representation for trees, similar to ES. Originally, Rothlauf [2] proposed only non-heuristic crossover and mutation operators. We extend this work in this paper and propose heuristic crossover operators which incorporate problem-specific knowledge.

NetDir crossover [2] creates two offspring $G_{o1} = (V, E_{o1})$ and $G_{o2} = (V, E_{o2})$ from two parental trees $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ in two steps. The first step splits the nodes V into two disjoint sets V_1 and V_2. All edges in E_1 connecting nodes either in V_1 or V_2 are inserted into G_{o1} or G_{o2}, respectively. Analogously, all edges in E_2 connecting nodes either in V_1 or V_2 are inserted into G_{o2} and G_{o1}, respectively. The second step completes the two offspring by iteratively inserting randomly selected parental edges until each offspring consists of $n - 1$ edges.

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

3) Differences: ES and NetDir are similar but 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 possibility 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.

B. Extending Crossover Operators

We bias the crossover operators to consider knowledge about edge orientation. By biasing the operators, 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, we introduce the modified weight w'_{ij} of an edge e_{ij}. It depends on the weight w_{ij} and orientation γ_{ij} of an edge and is defined as

$$w'_{ij} = \alpha w_{ij}/w_{max} + (1-\alpha)\gamma_{ij}/\gamma_{max}, \tag{4}$$

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 tree, we introduce w''_{ij}. We consider edge orientation only if the distance $d_{(ij),C}$ between an edge e_{ij} and the center C of a graph (see Fig. 1) exceeds a predefined value. It is calculated as

$$w''_{ij} = \begin{cases} w'_{ij} & \text{if } dist_{(i,j),C}/dist_{max} \geq \beta \\ w_{ij}/w_{max} & \text{else,} \end{cases} \tag{5}$$

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 lower 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 and we obtain the original ES crossover operator (see Sect. III-A1). For $\alpha < 1$, edge orientation influences the probabilities of edges being included in an offspring and edges pointing towards the center of a tree are preferred.

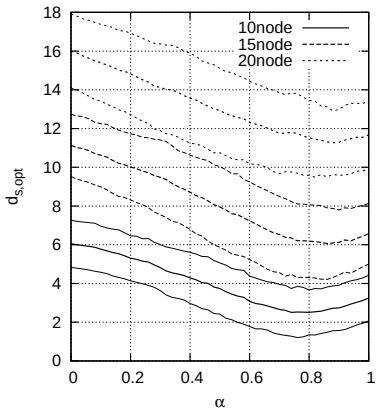


Fig. 3. Average distance $d_{s,opt}$ between trees T_s generated by a greedy selection strategy and optimal solutions T_{opt} over α for randomly generated OCST instances with 10, 15, and 20 nodes.

C. Balancing Weight and Orientation

This section neglects all crossover steps that do not consider edge weights (like transferring all common edges to an offspring in *-strategies) but focus only on edge-selection strategies. We examine how performance of a greedy selection strategy depends on α and β. Although the experiments explored all possible combinations of α and β, due to space restrictions, Figs. 3 and 4 present only the most promising combination. The greedy 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 selection strategy creates an MST.

The quality of a solution created by the greedy selection strategy is determined by how closely it approaches an optimal solution. We present results for 100 random OCST problems with 10, 15, and 20 nodes. Optimal solutions are determined according to Section II-B.

Figure 3 shows the average distance $d_{s,opt}$ between trees T_s created by a greedy selection strategy and optimal solutions T_{opt} over 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 selection strategy can create better solutions. Finally, with lower values of α, $d_{s,opt}$ increases. Considering only orientation ($\alpha = 0$) results in worse solutions than considering only distance weights ($\alpha = 1$).

Next, we show how the performance of a greedy selection strategy using w'' depends on β. Figure 4 shows the average distance $d_{s,opt}$ over 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 selection strategy creates MSTs. For $\beta = 0$, the resulting $d_{s,opt}$ are equivalent to the lowest distances found in Figure 3. For $\beta \approx 0.3$, better solutions more closely approaching optimal

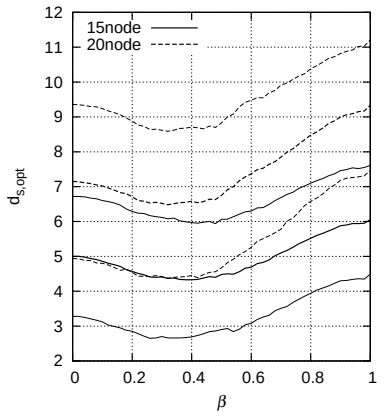


Fig. 4. Average distance $d_{s,opt}$ between trees T_s generated by a greedy strategy and optimal solutions T_{opt} over β for randomly generated OCST instances with 15 and 20 nodes. The greedy algorithm subsequently inserts edges into a tree starting with edges of low w''_{ij}.

trees can be obtained. Therefore, greedy selection strategies create better solutions if orientation is considered only for edges which are some distance from the tree center.

We recommend considering 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 $\beta = 0.3$.

IV. Performance of EAs using the extended operator

We compare the performance of EAs considering distance weights and edge orientation in edge-set and NetDir. We start with small problem instances where optimal (or near-optimal) solutions are determined as described in Section II-B and continue with larger problem instances with unknown optimal solutions. Following previous work [4], [28] we use randomly created OCST test instances.

A. Small Problem Instances

We use a basic steady-state EA with non-heuristic initialization and mutation [2], [4]. 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 lower 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. We present results for OCST instances of different sizes $n = \{10, 12, 14, 16, 18, 20\}$, where the optimal solutions are determined as described in Section II-B. 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 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 [4], [28].

Table I lists the percentage P_{suc} of runs that find 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. We show 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 lowest average costs $w(T_{best})$ are observed for $\alpha = 0.7$ and $\beta = 0.3$. Results for ES and NetDir are consistent.

Previous results [28] indicate that performance of heuristic ES is good if optimal solutions are similar to MSTs. Since optimal solutions for OCST problems 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. We expect that use of edge orientation as an additional selection criterion improves the performance of heuristic ES for larger $d_{opt,mst}$. Fig. 5 shows P_{suc} and the gap $\frac{w(T_{best})-w(T_{opt})}{w(T_{opt})}$ (in percent) over $d_{opt,mst}$ for 1000 random problem instances. We present results only for instances of size $n = 12$ and 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 low $d_{opt,mst}$. Since for these problem instances optimal solutions are only a few edges different from MSTs, 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 [28]. 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 Fig. 4, EA performance increases when neglecting edges next to the graph center ($\beta = 0.3$). Results for the 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 low n since the gap increases with larger n (see in Tables III and IV).

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

B. Larger Problem Instances

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

We use the same EA as in the previous experiments with a larger population size of 200. Since a higher number of

TABLE I
EA PERFORMANCE FOR SMALL PROBLEM INSTANCES

n		MST	Edge-Set RX	Edge-Set β = 0 α=1	Edge-Set β = 0 α=0.8	Edge-Set β = 0 α=0.7	Edge-Set β = 0 α=0.6	Edge-Set β = 0.3 α=0.8	Edge-Set β = 0.3 α=0.7	Edge-Set β = 0.3 α=0.6	NetDir RX	NetDir β = 0 α=1	NetDir β = 0 α=0.8	NetDir β = 0 α=0.7	NetDir β = 0 α=0.6	NetDir β = 0.3 α=0.8	NetDir β = 0.3 α=0.7	NetDir β = 0.3 α=0.6
demand uniformly distributed in]0,10]																			
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
demand Zipf distributed in [1,10]																			
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

TABLE II
NUMBER OF EVALUATIONS $eval$ FOR LARGER PROBLEM INSTANCES

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

evaluations improves EA performance, we also increase $eval$ with larger n (see Table II). We present results for OCST instances with $n = \{25, 50, 75, 100\}$. For each problem size, 100 random instances are created. Due to computational restrictions, we consider only 50 instances for $n = 75$ and 25 for $n = 100$. For each problem instance, we perform 20 independent EA runs on a dual core Intel processor with 2 GHz and 4 GB RAM running 64bit Linux.

For different n, we list $w(T_{best})$ using either ES (Table III) or NetDir (Table IV). Additionally, we show the standard deviations σ of $w(T_{best})$ and the average running time t_{cpu} (in seconds). 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 selection strategies considering 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 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 Fig. 6 show the average gap $\frac{w(T_{mst})-w(T_{best})}{w(T_{mst})}$ (in percent) between the best found solution T_{best} and an MST over n. A large gap indicates good EA performance. We have chosen the MST as reference since it is already a high-quality solution for OCST problems. 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 offspring's edges ($\alpha = 0.7$). Performance can be further improved if edges next to the graphs' centers are neglected ($\beta = 0.3$).

V. 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 tree's center occur with higher probability. Thus, EA performance can be systematically improved by biasing the search operators to favor such edges.

TABLE III
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

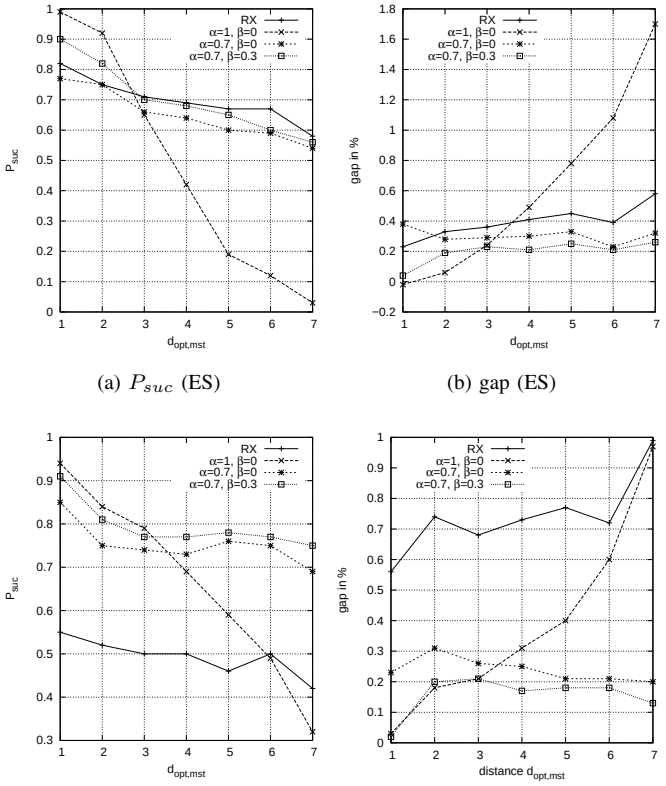
TABLE IV
EA PERFORMANCE FOR LARGE PROBLEM INSTANCES USING NETDIR

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

We exploit 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. We propose an extended crossover operator 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 tree 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 tree and have low distance weights. Considering both criteria results in a robust EA performance, and also problems where the optimal solution is quite different from MSTs can be solved. While we focus in this work on OCST problems, the basic approach is generally applicable to other Euclidean graph problems.

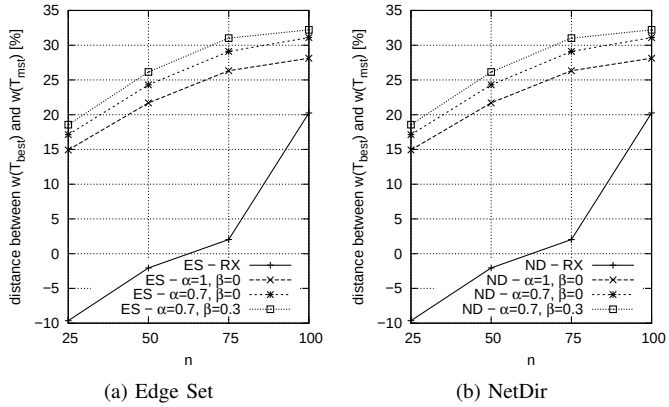
Future work will address what other problem-specific knowledge of tree problems can be used for designing high-quality EAs. While current approaches use only properties of edges, future work will analyze properties of tree structures.



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

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

Fig. 5. Edge set and NetDir performance using different edge-selection strategies for randomly generated OCST instances ($n = 12$). We show average success probability P_{suc} over $d_{opt,mst}$ (left) and average gap $\frac{w(T_{best})-w(T_{opt})}{w(T_{opt})}$ (in percent) over $d_{opt,mst}$ (right).



(a) Edge Set (b) NetDir

Fig. 6. EA performance using different crossover variants for randomly generated OCST instances. The plots show the average gap between the cost of the best found solution and the MST over the problem size n. The larger the gap, the higher EA's performance.

Other promising areas are problem-specific mutation and initialization operators for edge-set or NetDir which consider edge orientation.

REFERENCES

[1] T. C. Hu, "Optimum communication spanning trees." *SIAM Journal on Computing*, vol. 3, no. 3, pp. 188–195, 1974.

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

[3] P. Sharma, "Algorithms for the optimum communication spanning tree problem," *Annals of Operations Research*, vol. 143, no. 1, pp. 203–209, 2006.

[4] G. R. Raidl and B. A. Julstrom, "Edge sets: an effective evolutionary coding of spanning trees," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 225–239, 2003.

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

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

[7] C. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," in *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 1988, pp. 229–234.

[8] B. Y. Wu and K.-M. Chao, *Spanning Trees and Optimization Problems*. CRC Press, 2004.

[9] D. Peleg and E. Reshef, "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, 1998, pp. 670–681.

[10] B. Y. Wu, K.-M. Chao, and C. Y. Tang, "Approximation algorithms for some optimum communication spanning tree problems," *Discrete Appl. Math.*, vol. 102, no. 3, pp. 245–266, 2000.

[11] ——, "A polynomial time approximation scheme for optimal product-requirement communication spanning trees," *J. Algorithms*, vol. 36, no. 2, pp. 182–204, 2000.

[12] C. C. Palmer, "An approach to a problem in network design using genetic algorithms," Ph.D. dissertation, Polytechnic University, Troy, NY, USA, 1994.

[13] Y. Li and Y. Bouchebaba, "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, 2000, pp. 162–173.

[14] F. Rothlauf, D. E. Goldberg, and A. Heinzl, "Network random keys: a tree representation scheme for genetic and evolutionary algorithms," *Evolutionary Computaton*, vol. 10, no. 1, pp. 75–97, 2002.

[15] S.-M. Soak, "A new evolutionary approach for the optimal communication spanning tree problem," *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, vol. E89-A, no. 10, pp. 2882–2893, 2006.

[16] T. Fischer and P. Merz, "A memetic algorithm for the optimum communication spanning tree problem," in *Hybrid Metaheuristics*, T. Bartz-Beielstein, M. J. B. Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, Eds., vol. 4771. Springer, 2007, pp. 170–184.

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

[18] Li, "Random texts exhibit zipf's law-like word frequency distribution," *IEEE Transactions on Information Theory*, vol. 38, 1992.

[19] V. Poosala, "Zipf's law," University of Wisconsin, Tech. Rep., 1995.

[20] F. Rothlauf, *Design and Application of Metaheuristics*. Universität Mannheim, Habilitationsschrift, 2007.

[21] F. Rothlauf and D. E. Goldberg, "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*, S. Brave and A. S. Wu, Eds. Orlando, Florida, USA: Omni Press, 1999, pp. 238–244.

[22] G. R. Raidl and J. Gottlieb, "Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem," *Evolutionary Computation*, vol. 13, no. 4, pp. 441–475, 2005.

[23] H. Prüfer, "Neuer beweis eines satzes über permutationen," *Archiv für Mathematik und Physik*, vol. 27, pp. 742–744, 1918.

[24] S. Even, *Algorithmic Combinatorics*. New York: The Macmillan Company, 1973.

[25] B. A. Julstrom and G. R. Raidl, "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*. New York, NY, USA: ACM, 2002, pp. 547–552.
[26] F. Rothlauf, "On Optimal Solutions for the Optimal Communication Spanning Tree Problem," *Operations Research*, vol. 57, no. 2, pp. 413–425, 2009.
[27] G. Raidl, "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, 2000, pp. 104–111.
[28] F. Rothlauf, "On the Bias and Performance of the Edge-Set Encoding," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 486–499, 2009.
[29] A. Kershenbaum, *Telecommunications network design algorithms*. New York: McGraw Hill, 1993.
[30] B. A. Julstrom and G. R. Raidl, "Weight-biased edge-crossover in evolutionary algorithms for two graph problems," in *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2001, pp. 321–326.
[31] W. Steitz and F. Rothlauf, "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*. New York, NY, USA: ACM, 2008, pp. 563–570.