JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

Using Penalties instead of Rewards: Solving OCST Problems with Problem-Specific Guided Local Search

Wolfgang Steitz, Franz Rothlauf

Working Paper 01/2011
March 2011

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`

Using Penalties instead of Rewards: Solving OCST Problems with Problem-Specific Guided Local Search

Wolfgang Steitz & Franz Rothlauf
steitzw/rothlauf@uni-mainz.de
Department of Information Systems & Business Administration
Johannes-Gutenberg Universität Mainz, Germany

April 4, 2011

Abstract

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

1 Introduction

The optimal communication spanning tree (OCST) problem [5] is a common $\mathcal{NP}$-hard combinatorial optimization problem which seeks a spanning tree that satisfies given communication requirements and leads to minimal total costs. Researchers studied various exact, approximate and heuristic solution approaches for the problem. The current state-of-the-art approaches [11, 3, 12] are based on heuristics and metaheuristics, in particular evolutionary algorithms (EA).

Previous work [17, 18, 13] studied properties of OCST problems. They found that edges with low distance weights pointing towards the center of a graph occur in high-quality solutions with higher probability. Additionally, optimal solutions are star-like, nodes with a large node degree are near the tree center, and nodes with small node degrees are far away from the center. Furthermore, in optimal solutions, more than half of the nodes are leaf nodes with a node-degree of only one. There are several possibilities of how such a-priori knowledge can be exploited in the design of efficient metaheuristics for OCST problems [13]:

1. Initialize metaheuristic with solutions that have similar properties than high-quality solutions: Steitz and Rothlauf [18] presented a simple initialization method which creates solutions with the desired tree properties. The solutions are used in the initial population of a genetic algorithm, which improved the overall search performance.

2. Use a redundant representation that over-represents solutions with certain properties: Rothlauf [12] applied the idea of using redundant representations that over-represent high-quality

building blocks to the link-biased (LB) encoding. The LB encoding encodes trees similar to MSTs with higher probability. Experiments show that using the LB encoding improves search performance.

3. Use search operators that favor building blocks, similar to those of known good solutions: Examples for this approach are the heuristic crossover and mutation operators of the edge-set (ES) encoding [11, 14]. Both, crossover and mutation, favor low-weight edges when creating offspring solutions, which leads to an performance increase in comparison to the non-heuristic operators. Similarly, the search operators of ES can be extended to include edge weights *and* edge orientation [17].

4. Assign higher fitness values to solutions with similar characteristics than high-quality solutions: We are unaware of any approach for the OCST problem that modifies the fitness function according to characteristics of high-quality solutions.

An approach to dynamically change the objective function in order to lead heuristic search to promising areas is guided local search (GLS) [22, 23]. To apply GLS, it is necessary to define solution features and corresponding costs. Natural features are building blocks of a solution. For a graph problem, these are typically the edges. Corresponding costs can depend on the weight and orientation of edges. During the search process, some low-quality solution features are penalized which helps local search to escape local optima. In this paper, we apply GLS to the OCST problem and show how problem-specific knowledge can be used to improve the search performance by changing the evaluation function. The main findings of this paper are:

1. Problem-specific GLS outperforms state-of-the-art EAs using edge-sets although EAs also consider weights and orientation of edges.

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

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

The next section defines the OCST problem, gives an overview over existing solution strategies and summarizes the knowledge about high-quality solutions. Sect. 3 presents GLS and develops a problem-specific GLS approach for the OCST problem. Sect. 4 presents experimental results for different types of OCST instances. The paper ends with concluding remarks.

2 The OCST Problem

2.1 Definition

The optimal communication spanning tree (OCST) problem is a common combinatorial tree optimization problem, first described by Hu [5]. Given a collection of nodes and the distances and communications demands between them, we seek a tree that connects all nodes at minimum total communications cost, defined to be the sum over all pairs of nodes of the products of the distances and communications demands between them.

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

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

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

Like many other constrained graph problems, the OCST problem is $\mathcal{NP}$-hard [4]. Furthermore, the problem is $\mathcal{MAX}$ $\mathcal{SNP}$-hard and no polynomial-time approximation scheme exists, unless $\mathcal{NP} = \mathcal{P}$ [8]. Hu [5] presented two cases of the problem which are polynomial solvable. For the general case of the OCST problem, Ahuja and Murty [1] proposed an exact approach based on branch-and-bound. Contreras et al. [2] formulated the problem as an integer programming problem. However, both approaches only work for small problem instances. In addition, various approximation algorithms for the OCST problem have been developed [9, 24, 15], although, due to the $\mathcal{MAX}$ $\mathcal{SNP}$-hardness of the problem, the solution quality of such approximation algorithms is very limited. To overcome the limitations of exact and approximation algorithms, many heuristics, especially EAs have been developed [7, 6, 13, 11, 16, 3].

2.2 Properties of high-quality solutions

Rothlauf [12] studied properties of OCST problems and found that optimal solutions are biased towards minimum spanning trees (MST). The distances between optimal solutions and MSTs are significantly lower than the distances between optimal solutions and randomly generated solutions. Thus, by using optimization methods that are biased towards MST-like solutions, search performance can be increased. Heuristic variants of edge-sets [11, 10] make use of this fact by favoring edges with small distance weights.

Steitz and Rothlauf [17] analyzed the orientation of edges in high-quality solutions for Euclidean OCST instances and found that edges with small γ are over-represented in optimal solutions. These are edges pointing towards the graph's center. For example, for n=20, on average about 20% of all edges of optimal solutions have an angle $\gamma \leq 10$ whereas only about 4% of all edges have an angle $\gamma > 80$. The larger the angle γ, the lower is the probability that this edge is part of the optimal solution. Therefore, biasing search operators to favor edges with low weight *and* low edge orientation leads to an performance increase [17].

3 Guided Local Search

A major problem of local as well as evolutionary search algorithms is their inability to escape local optima. There exist a wide range of mechanism to avoid this problem, like Tabu lists (used in Tabu Search) or accepting worse solutions with some probability (used in simulated annealing). Guided local search (GLS), which was presented by Voudouris in 1997 [21, 23, 22], ensures diversification by dynamically changing the objective function by adding penalties based on the knowledge gained during the search. By dynamically modifying the fitness function, search does not get stuck in local optima but is forced to leave local optima and to explore other areas in the search space. Usually, the penalties consider information about previous search steps, but they can also incorporate prior knowledge about the problem. GLS is easy to implement and apply, as there are only few parameters.

GLS has been successfully applied to a wide range of combinatorial optimization problems, such as the traveling salesman problem [23], function optimization, partial constraint satisfaction, vehicle routing, and others [22]. This section outlines the functionality of the algorithm and demonstrates the design of a problem-specific GLS variant for OCST problems.

3.1 Guided Local Search Algorithm

GLS's mechanism to change the objective function is based on *solution features*, which represent solution properties. A solution feature is a non-trivial property of a solution and allows to distinct

between different solutions. If GLS reaches a local optima, the worst solution feature is penalized. Penalizing solution features alters the objective function and directs the search to other areas of the search space.

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

$$h(T) = f(T) + \lambda \cdot \sum_{i=1}^{M} p_i \cdot I_i(T), \tag{2}$$

where T is the tree to be evaluated, f is the original evaluation function with no modifications (see equation 1), M is the number of solution features, p_i is the penalty for solution feature i, and parameter λ controls the importance of the penalties. The indicator function $I_i(T)$ tells us whether feature i is part of a solution T, or not:

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

Tsang et al. [20] suggested to set the parameter λ according to the first found local optimum T_1 as

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

This allows us to set the parameter α independently of the problem instance [22, 20].

Algorithm 1 GuidedLocalSearch(V, E, w)

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

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

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

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

Then, the next iteration starts from the current local optimum. Since the evaluation function is altered, the current solution is not a local optimum any more and local search is able to escape

from the local optimum and explore new solutions. If the stopping condition, e.g. a given number of iterations, is reached, the algorithm terminates and returns the best found solution so far.

3.2 Solving the OCST problem using GLS

As mentioned by Voudouris [22], edges are proper solution features for graph problems. Therefore, we define the set of solution features equal to all edges E in the fully connected graph. Then, the number of solution features M that are part of a solution T is $n-1$. For random instances of OCST problems (see Sect. 4.1), where the distance weights w_{ij} are chosen randomly, proper feature cost c_{ij} are the distance weights w_{ij}.

For Euclidean OCST problem instances, where the nodes are set on a two-dimensional plane and the distance weights w_{ij} are the Euclidean distances between the nodes, the angle γ_{ij} denotes the orientation of edge e_{ij} ($\gamma_{ij} = 0$ if edge e_{ij} points towards the center of the graph and $\gamma_{ij} = 90$ if edge e_{ij} is perpendicular to the line connecting the middle of e_{ij} and the graph's center). Following Steitz and Rothlauf [17], the quality of an edge depends not only on its distance weight w_{ij} but also on its orientation γ_{ij}. Therefore, we distinguish between three possible types of cost c_{ij}:

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

$$c^o_{ij} = \begin{cases} \gamma^*_{ij} & dist^*_{(i,j),C} \geq 0.3 \\ w^*_{ij} & else. \end{cases} \tag{3}$$

3. If we consider distance weight *and* orientation, Steitz and Rothlauf [17] recommended to use

$$c^{ow}_{ij} = \begin{cases} 0.7w^*_{ij} + 0.3\gamma^*_{ij} & dist^*_{(i,j),C} \geq 0.3 \\ w^*_{ij} & else. \end{cases} \tag{4}$$

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

Following [17], edge orientation is only considered for edges that are some distance away ($dist^*_{(i,j),C} \geq 0.3$) from the graph's center. This is necessary, as edge orientation is meaningless for edges next to the graph center. In contrast to c^w_{ij} and c^o_{ij}, which only consider either distance weights or edge orientation, c^{ow}_{ij} considers both weight *and* orientation.

The local search LocalSearch(T_k) starts with T_k and iteratively examines a randomly selected neighboring tree. If a neighboring tree has a lower objective value h than the original solution, it replaces the original tree. The neighborhood of a tree consists of all trees that can be created by changing one edge. To generate neighboring trees, the insert-before-deletion operator from Raidl and Julstrom [11] is used. This local search operator chooses a random edge not yet included in the tree and inserts it, creating a cycle. To restore a feasible tree, the operator identifies the edges forming the cycle and removes one of those edges at random. Cycle detection is done using a depth-first search and the running time is $O(n)$. Raidl and Julstrom [11] also proposed a heuristic variant of the local search operator which prefers edges with lower weight when inserting new edges. The number of different neighbors of a tree depends on the tree's structure and varies between $(n-1)(n-2)$ and $\frac{1}{6}n(n-1)(n+1) - n + 1$. If there is no improvement for l consecutive search steps, the search process is likely stuck in a local optima. Then, the local search is stopped and the solution feature with highest $util_i$ is penalized.

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

The only GLS parameter requiring tuning is $\alpha \in]0, 1]$. For the OCST problem, we tried different settings and obtained best results for $\alpha = 0.3$. This parameter setting was also recommended by Tsang et al. [20] studying other combinatorial optimization problems.

Table 1: Maximum number of evaluations *eval* for different problem sizes

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

4 Experimental Results

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

4.1 Problem instances

We follow previous work [11, 13] and use randomly created OCST test instances. The real-valued demands r_{ij} are randomly created and uniformly distributed in $]0, 10]$. For the random instances, the real-valued distance weights w_{ij} are also chosen at random in the interval $[0, 10]$. For the Euclidean instances, the distance weights w_{ij} are calculated as the Euclidean distances between the nodes i and j which are randomly placed on a 10×10 2-dimensional grid.

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

For each problem type, we create instances of different size (n=10, 15, 25, 50, 75, 100). For every problem size, we create 1000 random instances. Due to long running times, we only create 100 instances for $n \geq 100$. Overall, we create 9.900 different OCST instances.

4.2 Experimental setup

We compare the GLS approach to a simple local search and an EA using edge-sets.

Local search (denoted LS) is equivalent to LocalSearch(T_0) (see Sect. 3.2), where T_0 is a random initial solution. In each search step it examines a random neighboring tree. If the new solution has lower cost than the original tree, it replaces the original tree. Like the other search methods, LS is terminated after *eval* fitness evaluations (see Table 1).

The steady-state EA uses edge-sets (ES) and the search operators proposed by Raidl and Julstrom [11]. The population size is set to $N = 200$. In each search step, a new solution T_{off} is created by recombination and mutation. We extend the original crossover operator, which only considers edge weights, and consider edge weights *and* edge orientations as proposed in [17] for Euclidean and clustered instances. Due to the bias of the crossover operator, solutions similar to MSTs are preferred. Mutation is applied after crossover with probability $p_m = 1/n$. The resulting offspring T_{off} is inserted into the population *pop* if it has lower cost than the worst solution in the population ($f(T_{off}) < min_{T \in pop} f(T)$). The initial population consists of random spanning trees. The EA terminates after *eval* fitness evaluations.

The functionality of GLS is described in Sect. 3.2. Initial solutions are created at random. For random problem instances, the cost of a solution feature (edge) are the edge weights ($c_{ij} = w_{ij}$). For Euclidean and clustered instances, we either use c^w_{ij} (edge weights), c^o_{ij} (edge orientation), or c^{ow}_{ij} (weight and orientation) as cost c_{ij} of a solution feature. The last 500 search steps of each GLS run are performed using the original cost function f to obtain a local optimum solution.

Each optimization run is terminated after *eval* fitness evaluations. Since a larger number of evaluations increases search performance, we also increase *eval* with larger n (see Table 1). For each OCST problem instance, ten optimization runs are performed. For small instances ($n = 10$) a branch-and-bound algorithm [1] is able to determine the optimal solution in reasonable time. Therefore, P_{suc} indicates the percentage of runs of a metaheuristic that finds the optimal solution.

Table 2: Performance comparison for OCST instances with random distance weights

n		ES	GLS 50	GLS 100	GLS 300	LS
10	P_{suc}	0.91	0.94	0.96	**0.98**	0.88
	$\mu(f(T))$	684.68	684.8	684.0	**683.71**	687.68
	σ	1.65	3.43	1.22	0.74	9.9
15	$\mu(f(T))$	1350.39	1338.24	1338.32	**1337.04**	1370.21
	σ	11.74	6.68	8.18	6.34	57.33
25	$\mu(f(T))$	2841.06	2758.55	2759.16	**2757.03**	3049.55
	σ	65.01	51.55	51.7	48.44	289.69
50	$\mu(f(T))$	8030.45	7517.0	7513.44	**7507.25**	9707.57
	σ	555.79	558.69	544.45	548.55	1438.86
75	$\mu(f(T))$	15779.15	**13705.38**	13789.65	13755.92	21290.05
	σ	1223.09	1106.94	1192.33	1103.11	3784.91
100	$\mu(f(T))$	23538.09	19103.72	**19090.51**	19152.27	36611.01
	σ	1806.64	1433.3	1452.21	1432.98	6974.1

4.3 Random instances

Table 2 presents results for the random test distances. We show the percentage P_{suc} of runs that find the optimal solution (only for $n = 10$), the mean cost $\mu(f(T))$ of the best solution at the end of a run and the corresponding standard deviation σ. We compare an EA using edge-sets (ES), local search (LS), and three different variants of GLS with $l = 100$, $l = 200$ or $l = 300$. l specifies the maximal number of non-improving search steps, until the current solution is declared a local optima and LocalSearch() is stopped.

We observe that ES and GLS clearly outperform the simple local search. The difference is smaller for small instances, since these instances are relatively easy to solve. Comparing GLS and ES indicates a high performance of the GLS approach. For all problem sizes, all GLS configurations are able to find solutions with significantly lower costs. The setting of l has no significant influence on the GLS performance, but setting $n = 100$ seems to be a good compromise.

Statistical tests confirm these observations. For $n \geq 50$ the differences between ES, LS and all GLS configurations are significant using a pairwise ranked t-test with an error level of $p < 0.0001$. Also, differences between the three GLS configurations are not significant with $p > 0.8$.

4.4 Euclidean instances

For Euclidean test instances, we study three different variants of GLS using different costs of the solutions features: The first variant (GLS-w) uses only edge weights c^w_{ij} as costs of solution features, the second one (GLS-o) uses only orientation c^o_{ij}, and the third one (GLS-ow) uses edge weight and edge orientation c^{ow}_{ij}. For all GLS configurations, $l = 100$. We analogously design the EA using edge-sets. We have heuristic variants, where the heuristic crossover operator [11, 17] uses either only edge weights c^w_{ij} (ES-w), only orientation c^o_{ij} (ES-o), or edge weights and edge orientation c^{ow}_{ij} (ES-ow). In addition, we have a naive non-heuristic variant [11], which does not use any problem-specific knowledge and is not biased towards low-weight edges (denoted as nES).

Table 3 presents the results. We show P_{suc} (only for $n = 10$), the mean cost $\mu(f(T))$ of the best solution at the end of a run and the corresponding standard deviation σ. Again, we observe that all GLS variants outperform ES for larger problem instances ($n \geq 25$). For smaller instances, ES is slightly better. Comparing the different ES configurations, we find that considering problem-specific knowledge for the crossover operator improves search performance. The more knowledge is integrated, the better are the results. ES-ow is significantly better than ES-o or ES-w. Furthermore, results are significantly better if we consider only edge weights (ES-w) in comparison to if we compare only edge orientation (ES-o). This confirms previous results, that edge weights are more important than edge orientation [18, 19]. Combining both properties of high quality edges namely low weights and pointing towards the center of a graph (ES-ow) leads to best results. A pairwise ranked t-test confirms these results. All p-values $p < 0.0001$, except

Table 3: Performance comparison for Euclidean OCST instances

n		nES	ES-w	ES-o	ES-ow	GLS-w	GLS-o	GLS-ow
10	P_{suc}	0.42	0.43	0.14	0.67	**0.74**	0.7	0.71
	$\mu(f(T))$	1499.06	1498.55	1535.0	**1492.37**	1494.96	1496.2	1495.49
	σ	12.84	7.2	20.28	3.42	11.34	12.9	11.81
15	$\mu(f(T))$	3745.5	3675.87	3881.14	**3617.82**	3622.12	3622.94	3624.59
	σ	90.21	32.06	77.71	15.77	45.04	45.25	46.9
25	$\mu(f(T))$	11866.67	10994.21	12336.88	10616.75	**10599.0**	10599.18	10599.2
	σ	530.62	149.33	439.71	76.89	198.46	188.21	188.34
50	$\mu(f(T))$	61310.54	48694.77	60946.8	45900.01	45578.73	45581.47	**45539.77**
	σ	5839.66	996.33	3273.73	647.76	1244.97	1226.11	1210.7
75	$\mu(f(T))$	133640.85	112189.55	136220.17	105026.63	103598.95	**103432.23**	103485.73
	σ	11269.05	2298.56	7053.38	1598.57	2657.63	2600.17	2514.29
100	$\mu(f(T))$	216765.26	199753.68	227468.29	187148.03	**185208.69**	185903.59	185479.76
	σ	12954.33	3914.58	8416.62	2404.58	4311.28	4886.46	4654.08

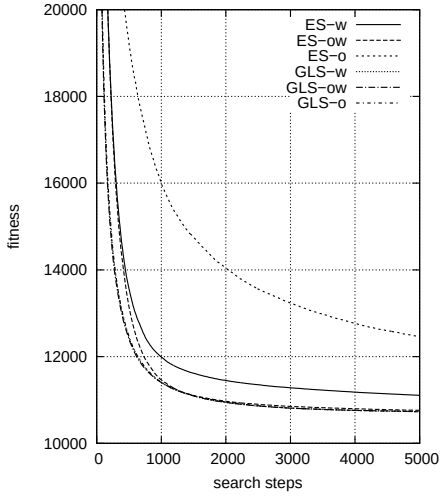


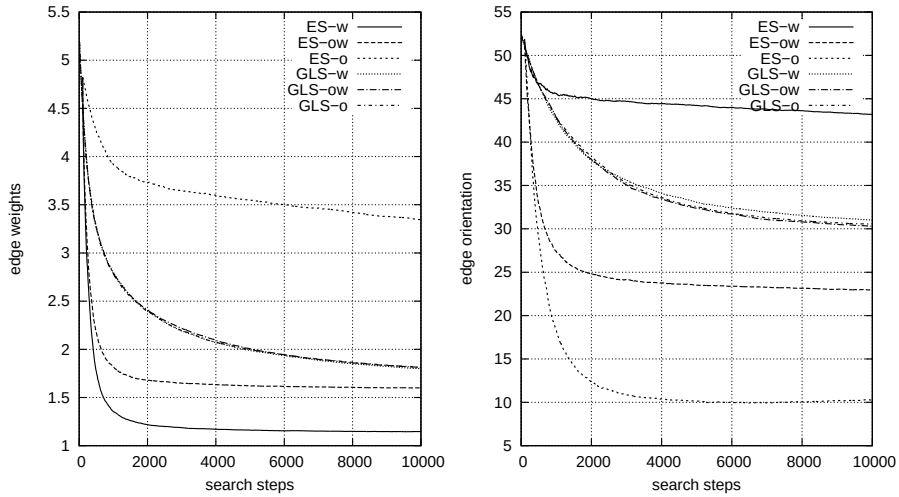
Figure 1: We show the average fitness over number of search steps for different variants and Euclidean instances ($n = 25$). All GLS variants show very similar behavior. In contrast, the ES variants differ significantly and ES variants considering both weight and orientation perform best.

for ES-ow vs. ES-w and $n = 25$.

The situation is different for GLS. The average fitness $\mu(f(T))$ at the end of a run shows no significant difference whether we use edge weights, edge orientation or both as costs of solution features. $p > 0.7$ using a pairwise ranked t-test for all problem sizes.

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

Figure 1 plots the average fitness of the current best solution over the number of search steps for $n = 25$. Results for other tree sizes are similar. The fitness values are averaged over all runs and all instances. For ES, there are no results for the first $M = 200$ evaluations, since these are necessary for evaluating the initial population. As expected, the GLS variants progress faster to high-quality solutions, since they mainly use intensification. Furthermore, the plots indicate



(a) edge weights (b) edge orientations

Figure 2: We show the average edge weight (a) and edge orientation (b) over the optimization progress for different configurations and Euclidean instances with $n = 50$.

a difference between the three ES variants. EAs considering orientation and weights perform better than considering only weights, which perform better than considering only orientation. In contrast, the progress of the GLS variants is almost identical.

As expected and as seen in the results, ES performance increases if more problem-specific knowledge is integrated. In contrast for GLS, considering additional knowledge has no influence. To further investigate the influence of problem-specific knowledge, we analyze how the average edge weights and edge orientation changes during a run. Figures 2 plot the average weight w_{ij} (Fig. 2a) and the average edge orientation γ_{ij} (Fig. 2b) of the edges in the current best found solution over the number of search steps.

For ES, the choice of problem-specific knowledge has a strong influence on the properties of the found solution. When considering only orientation (ES-o), an ES run finds good solutions with many edges that point towards the graph center (low γ; Fig. 2b). Edge weights are also reduced (Fig. 2a), but less in comparison to other variants or in comparison to the change of γ. When considering only edge weights (ES-w), we observe the opposite behavior: ES finds solutions with low-weight edges (Fig. 2a), which do not necessarily point towards the center of the graph (Fig. 2b). If we consider both weights and orientation (ES-ow), we are able to find solutions with low-weight edges that point towards the graph center, however less than when considering only edge weights (ES-w) or orientation (ES-o), respectively. In contrast, for GLS the choice of the cost of solution feature has no significant influence. All three variants (using either w_{ij}^{w}, w_{ij}^{o}, or w_{ij}^{ow}) show similar behavior. Having in mind that GLS finds better solutions than ES, we see that penalizing edges that have large weights or that do not point towards the center of a graph is a more promising strategy than favoring low-weight edges that point towards the tree's center.

4.5 Clustered Instances

Table 4 presents the results for the clustered instances. The experimental setup is identical to the experiment in the previous paragraphs. Again, GLS clearly outperforms ES for larger problem instances. Again, the different GLS configurations differ not significantly. Creating clustered instances has no influence on the analyzed optimization methods. This indicates a robust search performance for both ES and GLS.

Table 4: Performance comparison for clustered instances

n		nES	ES-w	ES-o	ES-ow	GLS-w	GLS-o	GLS-ow
10	P_{suc}	0.36	0.5	0.09	0.67	**0.75**	0.71	0.73
	$\mu(f(T))$	1852.51	1847.58	1914.55	**1843.52**	1845.43	1846.43	1846.02
	σ	15.17	6.35	30.51	3.84	10.59	12.09	11.07
15	$\mu(f(T))$	4426.9	4304.98	4719.31	4268.37	**4264.9**	4266.15	4265.63
	σ	93.44	29.04	125.0	15.09	36.51	36.53	35.46
25	$\mu(f(T))$	10671.15	9548.48	12268.15	9399.99	**9283.56**	9297.34	9292.31
	σ	425.13	101.57	724.89	83.56	85.5	78.04	87.96
50	$\mu(f(T))$	39083.92	29208.55	48010.75	28450.06	**27586.97**	27615.93	27598.45
	σ	3334.79	440.05	4163.42	304.68	236.72	235.1	227.61
75	$\mu(f(T))$	76018.9	59421.75	94933.55	58152.51	55970.74	55941.04	**55890.39**
	σ	8770.66	741.36	8740.25	628.72	529.1	482.07	460.58
100	$\mu(f(T))$	98187.44	90060.36	124934.51	89151.35	**86648.06**	86744.46	86666.78
	σ	5176.01	675.9	10703.97	762.81	633.14	600.82	653.28

5 Conclusions

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

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

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

References

[1] R. K. Ahuja and V. V. S. Murty. Exact and heuristic algorithms for the optimum communication spanning tree problem. *Transportation Science*, 21(3):163, 1987.

[2] I. Contreras, E. Fernández, and A. Marín. Lagrangean bounds for the optimum communication spanning tree problem. *TOP*, 2009.

[3] T. Fischer and P. Merz. A memetic algorithm for the optimum communication spanning tree problem. *Lecture Notes in Computer Science*, 4771:170, 2007.

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

[5] T. C. Hu. Optimum Communication Spanning Trees. *SIAM Journal on Computing*, 3(3):188–195, September 1974.

[6] Y. Li and Y. Bouchebaba. A new genetic algorithm for the optimal communication spanning tree problem. *Lecture Notes in Computer Science*, page 162–176, 2000.

[7] C. Palmer and A. Kershenbaum. An approach to a problem in network design using genetic algorithms. *Networks*, 26:151–163, 1995.

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

[9] D. Peleg and E. Reshef. Deterministic polylog approximation for minimum communication spanning trees. *Lecture Notes Comput Sci*, 1443:670–681, 1998.

[10] G. R. Raidl. An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *In Proceedings of 2000 IEEE International Conference on Evolutionary Computation*, volume pages, pages 43–48. ACM Press, 2000.

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

[12] F. Rothlauf. An Encoding in Metaheuristics for the Minimum Communication Spanning Tree Problem. *INFORMS Journal on Computing*, 21(4):575–584, February 2009.

[13] F. Rothlauf. On Optimal Solutions for the Optimal Communication Spanning Tree Problem. *Operations Research*, 57(2):413–425, 2009.

[14] F. Rothlauf. On the Bias and Performance of the Edge-Set Encoding. *IEEE Transactions on Evolutionary Computation*, 13(3):486–499, June 2009.

[15] P. Sharma. Algorithms for the optimum communication spanning tree problem. *Annals of Operations Research*, 143(1):203–209, 2006.

[16] S. Soak. A new evolutionary approach for the optimal communication spanning tree problem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 89(10):2882, 2006.

[17] W. Steitz and F. Rothlauf. Orientation matters: how to efficiently solve ocst problems with problem-specific EAs. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, page 563–570. ACM New York, NY, USA, 2008.

[18] W. Steitz and F. Rothlauf. New insights into the OCST problem: integrating node degrees and their location in the graph. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, page 357–364. ACM, 2009.

[19] W. Steitz and F. Rothlauf. Edge orientation and the design of problem-specific crossover operators for the ocst problem. *IEEE Transactions on Evolutionary Computation*, forthcoming, 2011.

[20] E. Tsang, P. Mills, and J. Ford. Extending guided local search – towards a metaheuristic algorithm with no parameters to tune. Technical Report 371, Department of Computer Science, University of Essex, 2002.

[21] C. Voudouris. *Guided Local search for combinatorial optimization problems*. Phd thesis, Department of computer science, University of Essex, Colchester,UK, July 1997.

[22] C. Voudouris. Guided local sarch. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 185–218. Springer, 2003.

[23] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113(2):469–499, March 1999.

[24] B. Y. Wu, C. K. M., and T. C. Y. A Polynomial Time Approximation Scheme for Optimal Product-Requirement Communication Spanning Trees. *Journal of Algorithms*, 36(2):182–204, 2000.