Invited paper

Determining hop-constrained spanning trees with repetitive heuristics

Manuela Fernandes, Luis Gouveia, and Stefan Voß

Abstract—The hop-constrained minimum spanning tree problem is the problem of determining a rooted spanning tree of minimum cost in which each path from the root node to any other node contains at most *H* hops or edges. This problem relates to the design of centralized tree networks with quality of service requirements (in telecommunications) and has a close relation with other tree problems. In this paper we investigate the adaptation of some well-known "repetitive" heuristics used for the capacitated minimum spanning tree problem to the hop-constrained minimum spanning tree problem and investigate some simple look ahead mechanisms for enhancing the quality of a savings heuristic. Computational results for a set of benchmark tests with up to 80 nodes are presented.

Keywords— hop-constrained spanning tree problem, meta-heuristics, pilot method, rollout method.

1. Introduction

The hop-constrained minimum spanning tree problem (HMST) is defined as follows. Given a graph $G = (V,E)$ with node set $V = \{0,1,\ldots,n\}$, edge set E as well as a cost c_e associated with each edge $e \in E$ and a natural number H, we wish to find a spanning tree T of the graph with minimum total cost such that the unique path from a specified root node, node 0, to any other node has no more than H hops (edges). The HMST is $\mathcal{NP}$-hard because it contains as a particular case (the case with $H = 2$) an $\mathcal{NP}$-hard version of the simple uncapacitated facility location problem (see [2, 8]). It has been shown by [16] that the HMST is even not in APX, i.e., the class of problems for which it is possible to have polynomial time heuristics with a guaranteed approximation bound. Related theoretical investigations can be found, e.g., in [14]. Special cases are considered, e.g., in [17].

The HMST models the design of centralized telecommunication networks with quality of service constraints. The root node represents the site of a central processor (computer) and the remaining nodes represent terminals that are required to be linked to the central processor. The hop constraints limit the number of hops (edges) between the root node and any other node and guarantee a certain level of service with respect to some performance constraints such as availability and reliability (see, e.g., [22]). Availability refers to the probability that all the transmission lines in the path from the root node to the terminal are working, and reliability corresponds with the probability that a session will not be interrupted by a link failure. In general, these probabilities decrease with the number of links in the path implying that paths with fewer hops have a better performance with respect to availability and reliability. Centralized terminal networks are also usually implemented with multidrop lines for connecting the terminals with the center. In such networks, node processing times dominate over queuing delays and fewer hops mean, in general, lower delays.

Lower bounding schemes for the HMST based on network-flow models have been suggested in [9, 10, 12]. A recent paper [3] summarizes these approaches and proposes new ones (column generation or cut generation) that are based on equivalent formulations, in terms of the corresponding linear programming relaxations. In fact, this survey paper clearly indicates that any type of heuristic solution is needed as optimal solutions for realistic problem sizes are currently out of reach. Lower bounds for an extended version of the problem, i.e., the Steiner tree problem with hop constraints can be found in [10, 19]. Interestingly enough, besides a cheapest insertion and the tabu search method described in [19] for the Steiner version of the problem, and Lagrangean heuristics proposed in [12], not much has been suggested for the HMST in terms of methods for obtaining good quality feasible solutions.

There are many problems that are related to the HMST. Besides the generalization to the Steiner version mentioned in the previous paragraph, we refer to the capacitated minimum spanning tree problem (CMST) (see, e.g., [20] and the references given there) or terminal layout problem that is usually described as the problem of determining a rooted spanning tree of minimum cost in which each of the subtrees off the root node contains at most a given number of K nodes. The connection of the HMST with the CMST is important due to several reasons. While the HMST involves a size (number of nodes or number of edges) constraint on each path leaving the root, the CMST involves a size constraint on the subtrees off the root. Thus, the HMST is an integer relaxation of the CMST and it is natural to assume that some ideas which work well for one problem may also work well for the other. One example of this is given by the fact that reasonably successful approaches for the two problems are based on so-called hop-indexed integer linear programming formulations (see, e.g., [3, 11]). In context of determining feasible solutions (which is more to the point in this paper), straightforward adaptations of minimum spanning tree algorithms for the two problems share a common disadvantage, namely that nodes farther from the root node cannot be linked to the closest nodes due to the additional constraints. Thus, it is natural to think that a savings heuristic that has been suggested and used for the CMST and which overcomes this disadvantage might

also be suitable for the HMST (this is explored in Section 2). This reasoning is extended in Section 3 to repetitive heuristics which are known to successfully improve the savings heuristics for the CMST.
Furthermore, we suggest another heuristic approach which is based on some look ahead feature (Section 4), i.e., a "repetitive" approach superimposed as a guiding process on some construction method. This idea relates to the pilot method developed in the early 90s for the Steiner tree problem in graphs (see [4, 5]). Lateron, similar ideas were developed under different names. The most famous one is the rollout method [1]. For a recent survey on the pilot method and related applications to many combinatorial optimization problems (see [21]).
Computational results in Section 5 show that repetitive methods can improve considerably over the results of the original heuristics, however, for the prize of considerably larger computation times. Furthermore, we observe that we can use the resulting objective function values as good upper cutoff values in available mixed integer programming solvers (such as CPLEX) for determining optimal solutions for previously unsolved problem instances of the HMST. We close with conclusions and directions for further research.

2. Construction heuristics

When trying to solve the HMST an obvious idea is to adapt and modify two algorithms which produce minimum unconstrained spanning trees in an undirected graph. One is the well known Prim [18] algorithm (which for simplicity, we do not describe in the paper). Starting the algorithm from node 0, it is quite easy to modify it in order to guarantee that it produces a feasible solution for the HMST. This algorithm behaves quite poorly for the HMST. In fact it exhibits a disadvantage that is also observed for a modified Prim algorithm for the CMST. As already mentioned in the previous section, the deficiency is that nodes far from the root cannot always be linked to closest nodes due to the additional (hop or capacity) constraints, leading to more expensive solutions.
Thus, we shall follow the literature on the CMST and propose a savings heuristics for the HMST that is quite similar to the well-known Esau-Williams algorithm (EW) [6]. This heuristic usually starts with the star solution (that is, every node linked directly to node 0). The best feasible change, i.e., the change which yields the largest savings, is performed. This is iteratively repeated until no savings can be obtained any more. To be more specific, let χ_i denote the cost of the edge linking the subtree containing node i to the root in the current solution. For each pair of nodes i and j in different subtrees we compute the savings s_{ij} of linking i with j, which is defined as $s_{ij} = \chi_i - c_{ij}$ if the operation of including edge (i, j) and removing the edge linking the subtree containing node i to the root leads to a feasible solution, and $s_{ij} = \infty$, otherwise. The best exchange is performed and the savings are recalculated for the next iteration. The process stops when no positive savings is available any more. The method could easily be applied to other feasible solutions and so it could be classified as improvement procedure, too.

3. Second order algorithms

Second order algorithms are an important class of algorithms following the idea of repetition (also considered to be a multi-pass heuristic). They iteratively apply a so-called first order procedure to different start solutions (where some edges are fixed to be included) and/or modified cost functions (where inhibitively high cost values have been assigned to some edges) thus forcing edges into or out of the solution. Savings procedures like the EW may be applied as "slave" procedures to generate or complete the solution. In each iteration, all possible modifications according to a given rule are checked. The best one is realized and the respective modifications are made permanent for the remaining iterations. The underlying principle is to investigate many good starting points through some greedy procedure and thereby to increase the possibility of finding a good solution on at least one repetition (see, e.g., [13, 15]).

3.1. General scheme

With the main objective of overcoming the typical greediness of simple constructive heuristics, we investigate the class of second order algorithms. As far as we know, this class of heuristics has first been suggested in [15] for the CMST. In such algorithms, several calls of a first order algorithm (which usually is a simple constructive heuristic) are made. At the beginning of each iteration of the algorithm, an adequate set of constraints is added to the problem and the first order algorithm is executed. In general, such constraints force or inhibit edges from being included in the solution. Small modifications on the first order algorithm permit us to obtain such modified solutions. For instance, to prevent an edge (p,q) from being included in the solution, we simply define $c_{pq} = M$, where $M > \max_{(i,j)\in E} c_{ij}$ before running the first order algorithm. For forcing an edge to be in the solution, one can define $c_{pq} = m$, with $m < \min_{(i,j)\in E} c_{ij}$.
A general outline of a second order algorithm is described as follows. Let HEUR denote any first order heuristic for the HMST (e.g., the EW) and let HEUR(S_1, S_2) denote its application to a modification of the data where the edges of S_1 are necessarily included in the solution determined by the algorithm and the edges in S_2 are excluded from it. Let $C(S_1, S_2)$ be the cost of this solution and let *COST* denote the objective function value of the best solution obtained so far. Finally, let *IT* denote the number of iterations of the overall method.

Second order algorithm (Basic scheme)

Initialization: $COST \leftarrow \infty$
Initialize S_1 and S_2
Loop: **for** $i = 1, IT$ **do**
Call HEUR(S_1, S_2)
if $C(S_1, S_2) < COST$ **then**
$COST \leftarrow C(S_1, S_2)$
Modify S_1 and / or S_2

Each iteration of the main loop corresponds to an execution of the first order algorithm after modifying adequately the data / the cost matrix according to the definition of the sets S_1 and S_2. After the IT iterations, $COST$ gives the cost of the best solution. The definition (and rule modification) of the sets S_1 and S_2 may also depend on previous solutions of the main algorithm. One example of such a rule is to prevent two nodes i and j of being in the same subtree (assuming that this has happened in the previous solution) [15]. In this way, we guarantee that the solution obtained in the current iteration is different from the one obtained in the previous iteration. This rule corresponds to inhibiting any edge that connects the two components containing the two nodes, respectively. Thus, S_2 is implicitly defined as containing all such edges.

One way of improving the basic scheme is to repeat it several times. Each time, the best solution obtained so far and the modifications (associated to the corresponding sets S_1 and S_2) are made permanent. To explain this improved version, let SP_1 (SP_2) denote the set of edges which are permanently included in (excluded from) the remaining solutions. Let HEUR(S_1, S_2, SP_1, SP_2) denote the first order algorithm modified in such way that the edges in the set $S_1 \cup SP_1$ must be included in the solution and the edges in the set $S_2 \cup SP_2$ must be excluded from the solution. The cost of such solution is denoted by $C(S_1, S_2, SP_1, SP_2)$. The parameter k identifies the iteration number of the outer loop and $IT(k)$ corresponds to the number of iterations of the inner loop at iteration k of the outer loop. In general, the algorithm stops when all the solutions obtained in the execution of an inner loop do not improve the best solution. In the scheme below, "COND" indicates a general stopping condition.

Second order algorithm (Improved scheme)

Initialization: $COST \leftarrow \infty$
$k \leftarrow 1$
$SP_1 \leftarrow \emptyset$
$SP_2 \leftarrow \emptyset$
Outer Loop: **while not** (COND) **do**
Initialize S_1 and S_2
Inner Loop: **for** $i = 1, IT(k)$ **do**
Call HEUR(S_1, S_2, SP_1, SP_2)
if $(C(S_1, S_2, SP_1, SP_2) < COST)$ **then**
$COST \leftarrow C(S_1, S_2, SP_1, SP_2)$
$A_1 \leftarrow S_1$
$A_2 \leftarrow S_2$
Modify S_1 and / or S_2
$SP_1 \leftarrow SP_1 \cup A_1$
$SP_2 \leftarrow SP_2 \cup A_2$
$k \leftarrow k + 1$

Clearly, different ways and rules to define the sets S_1 and S_2 lead to different second order algorithms. Some of those rules are specified below.

3.2. Inhibiting edges

In this section we only consider inhibitions and thus, the sets S_1 and SP_1 are empty. Several ways of defining the sets S_2 and / or SP_2 are given next.

Simple inhibition (I1). Each edge (not incident to the root) of the previous solution defines the set S_2 in each iteration of the inner loop. (In this case we have $IT(k) = n$ for all k.) The edge leading to the best solution is made permanent and included in SP_2.

Inhibition of pairs of edges (I2). This case is similar to simple inhibition, but now each pair of edges is considered to be included in S_2 (in each iteration). The main idea of this rule is to increase the number of solutions generated by the main body of the algorithm and thus, to increase the possibility of finding a better solution. However, it is clear that this version of the algorithm requires an increased computational time.

The original inhibit in each iteration of the outer loop of the general procedure asks for all edges of the previous solution to be inhibited (I1) or to check for all possible exclusions of two edges at a time (I2). In each iteration of I1 we have a linear number of calls of the first order algorithm and in I2 this number is quadratic.

3.3. Forcing edges

In this section we only consider edge inclusions and thus, the sets S_2 and SP_2 are empty. Several ways of defining the sets S_1 and / or SP_1 are possible.

Join. Here we follow [15] and consider the following two sets of edges:

$$JA_1 = \{(p,q) | c_{pq} = \min_{r=1,\ldots,n} c_{rq}, q = 1,\ldots,n\},$$

$$JA_2 = \{(p,q) | c_{pq} = \min_{r=1,\ldots,n} c_{rq} : c_{0r} \leq c_{0q}, q = 1,\ldots,n\}.$$

The set JA_1 includes for each node the minimum cost edge incident to it. The set JA_2 includes for node q the minimum cost edge incident to it such that the other endpoint is closer to the root node.

In each iteration of the inner loop, S_1 contains exactly one edge from the set $JA_1 \cup JA_2$. Thus, the inner loop will have at most $2n$ iterations (as there may be edges repeated in the two sets and each set has at most n edges). The edge leading to the best solution is made permanent and the algorithm continues.

General join. In this rule we follow the inhibit concept and try to use information from previous solutions to define candidate edges to be included in the next solution. Clearly, all edges not included in the previous solution can be considered as candidates. Thus, one such edge defines a possible set S_1 (edges leading to infeasible solutions when considered together with SP_1 are excluded – this also refers

to edges leading to cycles or edges leading to paths which are too long). However, such an approach is too time consuming as there are many edges to examine. Thus, we consider a variation of the method where we only examine the z least cost edges incident with each node (and not leading to infeasible solutions). In our computational results we shall consider $z = 4$ (this choice results from some experiments reported in [7]).

4. Looking ahead with repetition

Building on a simple greedy algorithm the *pilot method* [4, 5] is a meta-heuristic that builds primarily on the idea of looking ahead for each possible local choice (by computing a so-called pilot solution), memorizing the best result, and performing the according move. One may apply this strategy by successively performing a first order algorithm for all possible local steps (i.e., starting with all incomplete solutions resulting from adding some not yet included element at some position to the current incomplete solution). The look ahead mechanism of the pilot method is related to increased neighborhood depths as the pilot method exploits the evaluation of neighbors at larger depths to guide the neighbor selection at depth one.

Usually, it is reasonable to restrict the pilot process to a given evaluation depth. That is, the pilot method is performed up to an incomplete solution (e.g., partial assignment) based on this evaluation depth and then completed by continuing with a conventional cheapest insertion heuristic. In fact, the general second order algorithm described above is nothing else than a pilot method of depth 1 where the local choice is performed by excluding edges from a given solution, one at a time.

For this paper we use the idea of looking ahead as a motivation for a more general repetitive second order algorithm. As a simple look ahead mechanism motivated by the pilot method we modify I2 as follows (called ILA). In each iteration, for each edge to be inhibited, we calculate a new solution and use all possible edges of that new solution as possible candidates for inhibition. Based on the general stopping criterion imposed by any type of savings heuristic like the EW to stop if no more improvements are possible we have two observations why ILA might have and in fact has larger computation times than I2. First, as intermediate solutions need to be calculated it definitely needs additional time for performing these calculations. Second, as we proceed with the method as long as improvements are possible, there may be an increased number of iterations of the outer loop based on additional improvements found.

5. Computational results

For computational comparisons we use a set of HMST problem instances with 40 and 80 nodes that have previously been used in the literature (see, e.g., [3, 8–10, 12]). We distinguish instances with the root in the center of a rectangle in the Euclidean plane (called TC) and instances with the root in the corner of the rectangle (called TE).

In order to reduce the size of each instance, we have used a simple edge elimination test (see [9]). If $c_{ij} > c_{0j}$, then any optimal solution does not use edge (i, j) and if $c_{ij} = c_{0j} (i \neq 0)$, then there is an optimal solution without edge (i, j). This means that edge (i, j) can be eliminated whenever $c_{ij} \geq c_{0j}$. This edge elimination test is applied to every instance before applying the heuristics. Note that the test is much more effective when applied to instances TC rather than to instances TE. This means that the reduced instances TE are larger than the reduced instances TC suggesting that the TE instances will be much more difficult to solve than the remaining instances. A standard way for reporting numerical results for heuristics found in the CMST literature is to compare with the EW (see, e.g., many of the papers surveyed in [20]).

Computational results for $n = 40$ and $n = 80$ are presented in Tables 1 and 2. In both tables, the first column, denoted by Prob, identifies the problem instance while the second column gives the value of H. For $n = 40$ we know the optimal solutions of all problem instances and they are provided in the third column (OPT). In Table 2 we provide optimal objective function values for the problem instances with $n = 80$ as far as we have been able to compute them based on the CPLEX models of [12] together with the upper bounds taken from Table 2 as upper cutoff values. That is, using the results from our computational study allowed us to compute optimal objective function values that had not been previously known. For the cases with $n = 80$, in case that we do not know an optimal solution from the literature and also cannot provide them with the above mentioned CPLEX models, we provide best known lower bounds *LB* taken from solving the linear programming relaxation of a hop-indexed formulation (see [3, 12]). They are shown in the form (LB).

The next few columns provide the heuristics that we tested. EW gives the upper bound obtained with the Esau-Williams algorithm. I1 and I2 provide the solutions obtained by the inhibition algorithms as described above. That is, I1 represents simple inhibition of edges following [15] and I2 means the case of forbidding two edges at the same time in each iteration of the inhibition procedure. Column ILA gives the results of the simple look ahead modification (applying the inhibition procedure). The column denoted by J shows the results obtained by means of the join procedure, which represents the junction of edges also according to [15]. In column J4 we provide the results for the general join procedure presented above. That is, in this procedure the four cheapest edges incident with each node, except the root, that do not belong to the solution, are candidates. Under the same headings we also provide the CPU times in seconds of the respective procedures. All results are obtained on a Pentium IV, 2.8 GHz following some straightforward implementation. Note that the implementation is straightforward and does not strive to incorporate versatile data structures to improve the CPU times as the major fo-

Table 1

Computational results and CPU times for the instances with $n = 40$

Prob	H	OPT	Upper bounds						CPU times [s]					
			EW	I1	I2	ILA	J	J4	EW	I1	I2	ILA	J	J4
	3	609	653	**612**	**612**	**612**	622	621	0.0	0.8	6.1	14.7	1.5	4.7
TC-1	4	548	594	564	563	**555**	560	563	0.0	0.7	6.5	16.6	1.1	4.3
	5	522	561	**524**	**524**	**524**	**524**	**524**	0.0	0.4	4.3	10.7	1.1	3.6
	3	566	596	572	**566**	**566**	587	575	0.0	0.7	4.5	9.5	1.4	4.8
TC-2	4	519	552	521	521	**519**	521	**519**	0.0	0.4	4.4	10.9	1.2	4.3
	5	496	**497**	**497**	**497**	**497**	**497**	**497**	0.0	0.3	2.3	4.8	1.2	4.0
	3	580	631	586	592	**582**	592	586	0.0	0.8	7.1	18.9	1.2	5.0
TC-3	4	544	585	557	549	**547**	561	550	0.0	0.6	9.6	19.5	1.2	4.4
	5	516	547	**520**	**520**	**520**	**520**	**520**	0.0	0.4	5.9	12.5	1.1	3.9
	3	613	661	624	620	**614**	632	619	0.0	0.5	5.2	13.3	1.4	5.0
TC-4	4	557	601	567	565	**561**	564	564	0.0	0.5	5.7	11.9	1.2	4.6
	5	524	552	543	539	537	536	**529**	0.0	0.4	4.5	11.1	1.1	3.7
	3	599	627	**604**	**604**	**604**	605	**604**	0.0	0.5	4.4	11.3	1.2	4.8
TC-5	4	552	583	560	560	560	**556**	558	0.0	0.7	6.3	13.6	1.1	4.2
	5	522	548	526	526	**522**	526	530	0.0	0.5	4.3	5.9	1.0	3.9
	3	708	834	**709**	**709**	**709**	733	724	0.0	1.0	9.6	20.1	2.7	8.3
TE-1	4	627	686	650	650	650	637	**635**	0.0	0.7	6.7	14.1	2.4	8.2
	5	590	676	627	605	**597**	611	599	0.0	0.6	8.6	20.0	2.2	7.4
	3	710	824	736	736	**730**	744	738	0.0	0.7	7.1	14.6	2.7	9.0
TE-2	4	625	694	653	**625**	628	659	651	0.0	0.8	11.3	27.1	2.3	8.4
	5	581	643	601	599	**593**	599	601	0.0	0.7	7.1	17.4	2.1	7.4
	3	660	779	692	690	**677**	746	686	0.0	0.8	11.1	30.5	2.6	9.3
TE-3	4	581	661	600	600	600	611	**584**	0.0	0.5	4.3	9.1	2.4	7.5
	5	540	602	556	**552**	**552**	562	557	0.0	0.5	6.1	15.2	2.1	7.4
	3	722	779	736	736	**732**	771	743	0.0	0.7	6.8	17.9	2.7	8.6
TE-4	4	625	674	640	**626**	**626**	647	648	0.0	0.7	7.9	16.6	2.2	7.8
	5	585	630	608	597	**591**	594	594	0.0	0.5	8.1	19.2	1.9	7.1
	3	675	791	687	686	**675**	705	682	0.0	1.2	9.0	26.1	2.9	8.6
TE-5	4	614	677	628	621	621	**620**	**620**	0.0	0.5	11.9	21.8	2.5	8.1
	5	571	600	582	580	579	**578**	**578**	0.0	0.7	7.5	15.7	2.1	7.8

cus is solely to investigate whether the solution quality can be enhanced and to which extent. Compared to the literature some of our results are providing new best solutions for these benchmark instances (although we believe that further improvements are possible).

Our results show that the EW performs quite reasonably for the HMST. (We have also implemented the Prim algorithm but do not show the results as they are much worse compared to those of the EW.) Inhibition and join as some simple repetitive mechanism following the idea of applying second order (multi-pass) heuristics clearly improve over the EW results. This is pretty much in line with our expectation and experience gained with respect to the CMST. Also, while not always being the case, the inhibition approach seems to outperform the join approach with respect to the solution quality obtained. This observation is also in line with earlier results from [15] for the CMST. Nevertheless, the computation times for J seem to be higher than those for I1.

Besides the quadratic number of repetitive calls of the first order heuristic (EW) in I2 and ILA for each iteration of the outer loop when compared to I1, an increased solution quality may influence the overall stopping criterion. Therefore, a considerably increased time behavior with respect to the extended algorithms seems obvious and can be clearly taken from the tables. With respect to solution quality the picture seems to be clear in the sense that repetition can really be beneficial. That is, while not always leading to the overall best results the ILA procedure seems to be better or at least as good than the other approaches in most cases. The additional effort for J4 seems to pay compared to J, but J4 seems to outperform ILA only in some cases.

Table 2
Computational results and CPU times for the instances with $n = 80$

Prob	H	OPT	Upper bounds						CPU times [s]					
			EW	I1	I2	ILA	J	J4	EW	I1	I2	ILA	J	J4
	3	1072	1233	1116	1106	1101	**1099**	1106	0.0	44.0	804.3	2011.5	63.9	224.1
TC-1	4	981	1073	1003	**997**	**997**	1011	1001	0.0	40.9	776.0	1796.0	55.6	192.9
	5	922	994	942	930	**929**	948	938	0.0	26.3	511.5	1208.0	48.1	167.7
	3	1054	1183	1115	1088	**1080**	1096	1086	0.0	22.8	801.3	1963.2	56.2	211.6
TC-2	4	967	1070	988	986	**982**	1006	994	0.0	45.9	864.5	2043.2	54.5	171.5
	5	918	993	930	933	**925**	938	936	0.0	33.6	664.1	1307.1	43.6	149.8
	3	1068	1169	1100	1092	**1091**	1100	1098	0.0	40.4	1016.7	2168.6	61.5	230.7
TC-3	4	968	1056	998	986	**982**	1000	997	0.0	36.4	807.6	1941.3	57.7	177.4
	5	912	978	924	920	**916**	930	928	0.0	26.2	641.0	1323.3	44.1	148.4
	3	1071	1168	1114	1110	1104	1111	**1097**	0.0	29.6	698.1	1909.5	65.9	213.1
TC-4	4	968	1049	990	**984**	986	1001	992	0.0	39.9	872.7	1730.3	55.7	184.6
	5	912	980	930	**916**	918	939	926	0.0	33.8	588.6	1399.4	48.1	162.6
	3	1235	1389	1288	**1262**	1268	1297	1274	0.0	21.9	848.3	1271.3	69.9	234.2
TC-5	4	1094	1232	1135	1134	**1126**	1134	1142	0.0	27.3	630.7	1257.1	58.6	206.1
	5	1037	1107	**1052**	**1052**	**1052**	1065	1059	0.0	25.8	426.4	1066.7	48.4	179.8
	3	(1743)	2321	2022	1965	1957	2003	**1928**	0.1	56.2	1044.6	2885.0	123.4	403.1
TE-1	4	(1462)	1953	1698	1666	**1631**	1708	**1631**	0.0	42.7	966.8	2675.1	110.0	372.1
	5	(1349)	1753	1508	**1500**	**1500**	1525	1516	0.0	27.3	601.0	1443.2	90.9	323.2
	3	(1669)	2403	1948	1926	**1845**	2120	1864	0.1	48.2	1082.7	2936.5	162.4	462.6
TE-2	4	(1413)	1814	1636	1548	**1542**	1635	1627	0.0	43.6	954.9	2364.3	133.6	407.6
	5	(1297)	1528	1451	1447	**1441**	1460	1445	0.0	22.2	472.8	1179.0	98.1	345.3
	3	(1672)	2364	1955	1872	**1864**	1960	1893	0.1	37.5	1158.4	2497.8	154.3	452.2
TE-3	4	(1432)	1816	1670	1625	1614	1656	**1601**	0.0	22.5	701.8	1667.1	118.3	365.4
	5	(1308)	1582	1506	1427	**1411**	1429	1426	0.0	22.6	446.5	1141.1	96.5	322.2
	3	(1679)	2409	1900	1864	**1843**	1954	1925	0.1	62.7	888.9	3024.1	141.2	448.7
TE-4	4	(1420)	1967	1704	**1631**	1665	1727	1665	0.1	31.2	1327.0	1831.8	136.5	422.8
	5	(1313)	1638	1499	1490	**1458**	1530	1464	0.0	31.6	981.5	1494.1	111.2	333.9
	3	(1692)	2478	1933	1909	1898	2017	**1892**	0.1	35.8	1435.0	2670.1	137.6	458.5
TE-5	4	(1447)	1775	1643	**1614**	**1614**	1647	1629	0.0	26.1	598.1	1344.3	105.9	387.1
	5	(1325)	1687	1494	1472	**1456**	1490	1458	0.0	10.3	441.1	1226.2	89.0	317.1

6. Conclusions and future research

To conclude with we have seen that the repetition mechanism within a second order algorithm may be worth investigating with respect to solution quality while the computation times are largely increasing. A simple look ahead feature may enhance this picture. With that it seems that results for so-called capacitated tree problems (naming the CMST and the HMST as such) using second order algorithms may be in line with each other. That is, once other capacitated tree problems are considered the use of second order algorithms may be a natural choice within future research.

References

[1] D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu, "Rollout algorithms for combinatorial optimization", *J. Heur.*, vol. 3, pp. 245–262, 1997.

[2] G. Dahl, "The 2-hop spanning tree problem", *Oper. Res. Lett.*, vol. 23, pp. 21–26, 1998.

[3] G. Dahl, L. Gouveia, and C. Requejo, "On formulations and methods for the hop-constrained minimum spanning tree problem", in *Handbook of Optimization in Telecommunications*, M. G. C. Resende and P. M. Pardalos, Eds. New York: Springer, 2006, pp. 493–515.

[4] C. W. Duin and S. Voß, "Steiner tree heuristics – a survey", in *Operations Research Proceedings 1993*, H. Dyckhoff, U. Derigs, M. Salomon, and H. C. Tijms, Eds. Berlin: Springer, 1994, pp. 485–496.

[5] C. W. Duin and S. Voß, "The pilot method: a strategy for heuristic repetition with application to the Steiner problem in graphs", *Networks*, vol. 34, pp. 181–191, 1999.

[6] L. R. Esau and K. C. Williams, "A method for approximating the optimal network", *IBM Syst. J.*, vol. 5, no. 3, pp. 142–147, 1966.

[7] M. Fernandes, "Estudo de limites inferiores e superiores para árvores com restrições de salto". Master thesis, Centro de Investigação Operacional, University of Lisbon, 2002 (in Portuguese).

[8] L. Gouveia, "Using the Miller-Tucker-Zemlin constraints to formulate a minimal spanning tree problem with hop constraints", *Comput. Oper. Res.*, vol. 22, pp. 959–970, 1995.

[9] L. Gouveia, "Multicommodity flow models for spanning trees with hop constraints", *Eur. J. Oper. Res.*, vol. 95, pp. 178–190, 1996.

[10] L. Gouveia, "Using variable redefinition for computing lower bounds for minimum spanning and Steiner trees with hop constraints", *INFORMS J. Comp.*, vol. 10, pp. 180–187, 1998.

[11] L. Gouveia and P. Martins, "A hierarchy of hop-indexed models for the capacitated minimum spanning tree problem", *Networks*, vol. 35, pp. 1–16, 2000.

[12] L. Gouveia and C. Requejo, "A new lagrangean relaxation approach for the hop-constrained minimum spanning tree problem", *Eur. J. Oper. Res.*, vol. 132, pp. 539–552, 2001.

[13] J. P. Hart and A. W. Shogan, "Semi-greedy heuristics: an empirical study", *Oper. Res. Lett.*, vol. 6, pp. 107–114, 1987.

[14] R. Hassin and A. Levin, "Minimum spanning tree with hop restrictions", *J. Algor.*, vol. 48, pp. 220–238, 2003.

[15] M. Karnaugh, "A new class of algorithms for multipoint network optimization", *IEEE Trans. Commun.*, vol. COM-24, no. 5, pp. 500–505, 1976.

[16] P. Manyem and M. Stallmann, "Some approximation results in multicasting", North Carolina State University, 1996.

[17] L. Monnot, "The maximum *f*-depth spanning tree problem", *Inform. Proces. Lett.*, vol. 80, pp. 179–187, 2001.

[18] R. Prim, "Shortest connection networks and some generalizations", *Bell Syst. Tech. J.*, vol. 36, pp. 1389–1401, 1957.

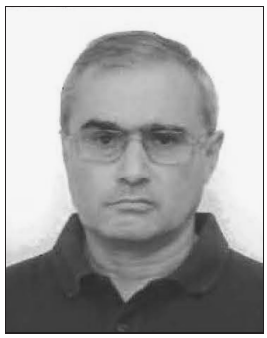
[19] S. Voß, "The Steiner tree problem with hop constraints", *Ann. Oper. Res.*, vol. 86, pp. 321–345, 1999.

[20] S. Voß, "Capacitated minimum spanning trees", in *Encyclopedia of Optimization*, C. A. Floudas and P. M. Pardalos, Eds. Dordrecht: Kluwer, 2001, vol. 1, pp. 225–235.

[21] S. Voß, A. Fink, and C. Duin, "Looking ahead with the pilot method", *Ann. Oper. Res.*, vol. 136, pp. 285–302, 2005.

[22] K. Woolston and S. Albin, "The design of centralized networks with reliability and availability constraints", *Comput. Oper. Res.*, vol. 15, pp. 207–217, 1988.

Manuela Fernandes, born 1969 in Castelo Branco, Portugal, is Assistant Lecturer of the School of Technology at the Polytechnic Institute of Tomar. Previous positions include Assistant Lecturer of the School of Technology at the Polytechnic Institute of Guarda from 1996 to 1999. She holds a B.Sc. degree in mathematics from the University of Coimbra and a M.Sc. degree from the University of Lisbon. Her main interests are heuristics and meta-heuristics for integer programming problems and networks with applications on telecommunications.
e-mail: manuela.fernandes@aim.estt.ipt.pt
School of Technology
Polytechnic Institute of Tomar
Quinta do Contador
Estrada da Serra
2300-313 Tomar, Portugal



Luis Gouveia, born 1957 in Mozambique, is Associate Professor at the Faculty of Sciences, University of Lisbon and Coordinator of the Operations Research Center at the Faculty of Sciences. He holds degrees in applied mathematics (diploma) from the University of Lisbon and a Ph.D. in operations research. His current research interests are in network design, ILP model reformulation and telecommunications. He is author of numerous papers in various journals. Luis Gouveia serves on the editorial board of some journals including being Associate Editor of "Networks" and "Computers & OR". He is frequently organizing workshops and conferences.
e-mail: legouveia@fc.ul.pt
Faculty of Sciences
University of Lisbon
Cidade Universitária, Bloco C6
Campo Grande, 1749-016 Lisboa, Portugal

Stefan Voß, born 1961 in Hamburg, Germany, is Professor and Director of the Institute of Information Systems at the University of Hamburg. Previous positions include full Professor and Head of the Division of Business Administration, Information Systems and Information Management at the University of Technology Braunschweig, Germany, from 1995 up to 2002. He holds degrees in mathematics (diploma) and economics from the University of Hamburg and a Ph.D., and the habilitation from the University of Technology Darmstadt. His current research interests are in quantitative/information systems approaches to supply chain management and logistics as well as public mass transit and telecommunications. He is author and co-author of several books and numerous papers in various journals. Stefan Voß serves on the editorial board of some journals including being Editor of "Annals of Information Systems", Associate Editor of "INFORMS Journal on Computing" and Area Editor of "Journal of Heuristics". He is frequently organizing workshops and conferences. Furthermore, he is consulting with several companies.
e-mail: stefan.voss@uni-hamburg.de
Institute of Information Systems
University of Hamburg
Von-Melle-Park 5
D-20146 Hamburg, Germany