Variable Neighborhood Search and Ant Colony Optimization for the Rooted Delay-Constrained Minimum Spanning Tree Problem

Mario Ruthmair and Günther R. Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
{ruthmair,raidl}@ads.tuwien.ac.at
http://www.ads.tuwien.ac.at

Abstract. The rooted delay-constrained minimum spanning tree problem is an NP-hard combinatorial optimization problem arising for example in the design of centralized broadcasting networks where quality of service constraints are of concern. We present two new approaches to solve this problem heuristically following the concepts of ant colony optimization (ACO) and variable neighborhood search (VNS). The ACO uses a fast construction heuristic based on node delays and local improvement exploiting two different neighborhood structures. The VNS employs the same neighborhood structures but additionally applies various kinds of shaking moves. Experimental results indicate that both metaheuristics outperform existing approaches whereas the ACO produces mostly the best solutions.

1 Introduction

When designing a communication network with a central server broadcasting information to all the participants of the network, some applications, such as video conferences, require a limitation of the maximal delay from the server to each client. Beside this delay-constraint minimizing the cost of establishing the network is in most cases an important design criterion. In another example we consider a package shipment organization with a central depot guaranteeing its customers a delivery within a specified time horizon. Naturally the organization aims at minimizing the transportation costs but at the same time has to hold its promise of being in time.

These network design problems can be modeled using a combinatorial optimization problem called *rooted delay-constrained minimum spanning tree (RDCMST) problem*. The objective is to find a minimum cost spanning tree of a given graph with the additional constraint that the sum of delays along the paths from a specified root node to any other node must not exceed a given delay-bound.

More formally, we are given an undirected graph $G = (V, E)$ with a set V of n nodes, a set E of m edges, a cost function $c : E \to \mathbb{R}_0^+$, a delay function

R. Schaefer et al. (Eds.): PPSN XI, Part II, LNCS 6239, pp. 391–400, 2010.

$d : E \to \mathbb{R}^+$, a fixed root node $s \in V$, and a delay-bound $B > 0$. An optimal solution to the RDCMST problem is a spanning tree $T = (V, E'),\ E' \subseteq E$, with minimum cost $c(T) = \sum_{e \in E'} c(e)$, satisfying the constraints:

$$\sum_{e \in P(s,v)} d(e) \leq B,\ \forall v \in V,$$

where $P(s, v)$ denotes the unique path from root s to node v.

The RDCMST problem is $\mathcal{NP}$-hard because already the special case with $d(e) = 1,\ \forall e \in E$, called *hop-constrained minimum spanning tree problem*, is $\mathcal{NP}$-hard [1].

The rest of the paper is organized as follows: In Section 2 existing approaches are discussed, Section 3 introduces some helpful preprocessing steps to reduce the problem size, Section 4 describes a metaheuristic method based on general variable neighborhood search, Section 5 presents a second approach based on ant colony optimization, Section 6 discusses test results, and Section 7 concludes the article.

2 Previous Work

Exact approaches to the RDCMST problem have been examined by Gouveia et al. in [2] based on the concept of constrained shortest paths utilized in column generation and Lagrangian relaxation methods. A flow-based reformulation of the problem on layered acyclic graphs is applied to reduce the RDCMST problem to the well-known and intensively examined *minimum Steiner tree problem* [3]. Since the size of the layered graph and therefore the efficiency of the according model heavily depends on the delay-bound B this approach can in practice only be used for instances with a reasonably small number of possible discrete edge delay values. Furthermore all these methods can only solve small instances with significantly less than 100 nodes to proven optimality in reasonable time when considering complete graphs.

A constructive heuristic approach based on Prim's algorithm to find a minimum spanning tree [4] is described by Salama et al. in [5]. A general problem of this heuristic especially on Euclidean instances is the fact that the nodes in the close surrounding of the root node are typically connected rather cheaply, but at the same time delay is "wasted", and many distant nodes can later only be linked by rather expensive edges. The stricter the delay-bound the more this drawback will affect the costs negatively. This fact led Ruthmair et al. [6] to a more de-centralized approach by applying the basic concept of Kruskal's minimum spanning tree algorithm [7] to the RDCMST problem. Two metaheuristics based on GRASP and variable neighborhood descent improve the constructed solution in [6].

There are many recent publications dedicated to the *rooted delay-constrained minimum Steiner tree problem* which is a generalization of the RDCMST problem. In this variant only a subset of the nodes has to be reached within the given

delay-bound, the other nodes can optionally be used as intermediate (Steiner) nodes. Several metaheuristics have been applied to this variant, such as GRASP [8,9], path-relinking [10] and variable neighborhood descent [11]. Also exact methods based on Integer Linear Programming have been explored, e.g. Leggieri et al. [12] describe a node-based formulation using lifted Miller-Tucker-Zemlin inequalities.

3 Preprocessing

The following rules are applied in a preprocessing phase in order to possibly reduce the instance graph without affecting optimal solutions.

3.1 Infeasible Edges

In the following cases an edge $e = (i, j) \in E$ cannot be part of a feasible solution so discarding it safely reduces the search space.
(a) Obviously all edges $e \in E$ having a delay $d(e)$ higher than the bound B can be discarded immediately.
(b) Edges $e = (i, j) \in E$ which would exceed the bound in all possible trees can also be removed from the graph [12] if satisfying these conditions:

$$d_{\min}(s, i) + d(e) > B \quad \wedge \quad d_{\min}(s, j) + d(e) > B,$$

whereas minimum delays $d_{\min}(s, v) := \min_{P(s,v)} \sum_{e \in P(s,v)} d(e)$, $\forall v \in V$, are calculated a priori by Dijkstra's shortest path algorithm [13] applied on the delays. Both preprocessing tests can be done on all edges E in time $\mathcal{O}(m + n \log n)$ including the calculation of d_{min} values.

3.2 Suboptimal Edges

Suboptimal edges can be part of a feasible solution but may not appear in an optimal solution, so discarding them safely prunes the solution space.

Comparison to Root Edges. Applying the arc elimination test presented in [14] to the RDCMST problem results in the removal of edge $e = (i, j) \in E$ if edges (s, i) and (s, j) exist and the following conditions hold:

$$c(s, j) \leq c(e),\ d(s, j) \leq d_{min}(s, i) + d(e) \quad \wedge$$
$$c(s, i) \leq c(e),\ d(s, i) \leq d_{min}(s, j) + d(e)$$

This preprocessing rule can be helpful for rather dense or complete graphs but in sparse graphs only few edges are typically discarded, mainly because of the small out-degree of the root node. Searching those edges takes time $\mathcal{O}(m)$ if d_{min} values are already known.

Extension to Arbitrary Triangles. Considering any triangle in the original graph consisting of edges $e_1 = (v_1, v_2),\ e_2 = (v_2, v_3),\ e_3 = (v_3, v_1)\ \in\ E$, edge e_1 cannot appear in an optimal solution if:

$$c(e_1) > c(e_2) + c(e_3)\ \wedge\ d(e_1) \geq d(e_2) + d(e_3)$$

If only the second part holds and $c(e_1) = c(e_2) + c(e_3)$, then edge e_1 can be part of an optimal solution but there has to be at least one other optimal solution with $e_1 \notin E'$ and we therefore can also remove it. This preprocessing step can be done in time $\mathcal{O}(mn)$.

Extension to Arbitrary Paths. The last preprocessing step can be further extended in the following way: Edge $e = (i, j)$ cannot appear in an optimal solution if there exists a path $P(i, j) \in E \backslash \{e\}$ satisfying the following conditions:

$$c(e) > \sum_{e' \in P(i,j)} c(e') \quad \wedge \quad d(e) \geq \sum_{e' \in P(i,j)} d(e')$$

Similarly if the second part is met and $c(e) = \sum_{e' \in P(i,j)} c(e')$, then edge e can be part of an optimal solution but there has to be at least one other optimal solution with $e \notin E'$. Applying this preprocessing test reduces to solving the constrained shortest path problem. This problem is $\mathcal{NP}$-hard but approximable by an FPTAS which implicates the existence of an exact pseudo-polynomial algorithm. An efficient dynamic programming approach customized to the RDCMST problem was presented by Gouveia et al. [2]. Nevertheless finding a delay-constrained shortest path runs in time $\mathcal{O}(mB)$ which extends to $\mathcal{O}(m^2 B)$ for the complete preprocessing test.

Especially the last two preprocessing steps must be used with caution. When having a limited runtime (as in our tests in Section 6) the preprocessing phase could dominate or even use up the whole time for large instances. So preprocessing can also be counterproductive and finally worsen solution quality.

4 General Variable Neighborhood Search

For constructing a feasible starting solution we use the Kruskal-based heuristic from [6]. To improve the quality of this solution we apply the metaheuristic framework *general variable neighborhood search* (GVNS) as introduced by Hansen et al. [15]. In each iteration the so far best solution is perturbed by shaking and then improved by an embedded *variable neighborhood descent* (VND).

4.1 Variable Neighborhood Descent

The VND used here is the one introduced in [6]. It performs a local search switching between two neighborhood structures: *Edge-Replace* and *Component-Renew*.

A move in the Edge-Replace neighborhood removes an edge and connects the resulting two components in the cheapest feasible way. A neighborhood search is done by next improvement considering the edges in decreasing cost order until a local optimum is reached, running in time $\mathcal{O}(nm)$.

A Component-Renew move also deletes an edge, but completely dissolves the component which is now separated from the root node; it then re-adds the individual nodes by applying Prim's algorithm [4] respecting feasibility. In some cases not all nodes can be added due to the delay-bound. These remaining nodes are again joined to the root component by shortest delay paths, dissolving created cycles. A neighborhood search is done in a similar way following a next improvement strategy considering the edges in decreasing cost order until a local optimum is reached, running in time $\mathcal{O}(n^3)$.

4.2 Shaking

Three different kinds of shaking moves are used in the GVNS framework. The algorithm always chooses one at random with equal probabilities:
(a) Replacing a random edge by another feasible randomly chosen edge not violating the delay constraint and tree structure.
(b) Adding the shortest delay path to a random vertex (see Section 3.1).
(c) Adding the delay-constrained least cost path to a random vertex (see Section 3.2).

To ensure feasibility of the solution two issues have to be considered: Firstly the last two shaking moves could possibly cause cycles which have to be dissolved, and secondly the delay-bound in the third move can be set at most to the global delay-bound reduced by the maximal delay in the subtree of the randomly chosen vertex. The number of shaking moves performed in one iteration is $\lceil |V| * sr \rceil$, where $sr \in (0, 1]$ is called shaking rate. This shaking rate is either set to a fixed value or dynamically changed in the search process. In the latter case sr is initialized with 0.01, increased by 0.01 when no better solution could be found in an iteration, limited from above by 0.3, and reset again to 0.01 in case of an improvement.

5 Ant Colony Optimization

We apply the concept of the $\mathcal{MAX} - \mathcal{MIN}$ Ant System (MMAS) from Stützle et al. [16] to our problem implementing the following key features:
(a) Only a single ant is allowed to deposit pheromones at the end of an iteration, either the best ant of the iteration or the globally best one, focusing the search to the best solutions found.
(b) Pheromone values are limited to an interval $[\tau_{min}, \tau_{max}]$ preventing a stagnation of the search.
(c) The initial pheromone values are set to τ_{max} leading to a high diversification at the beginning of the search.

Combining all these features provides both intensification and diversification throughout the search process, which is essential for a well-performing metaheuristic.

5.1 Pheromone Values

Pheromone values $\tau_{v,d} \in [\tau_{\min}, \tau_{\max}]$ are defined for each node $v \in V$ in combination with any delay $d \in (0, B]$ it might have. Here, d denotes the sum of delays of all edges on the path from the root to node v. The root s by definition always has a node delay 0 and therefore has no pheromone values associated.

Notice that in general delays are real values and therefore pheromone values do not form a classical finite matrix. However, when considering one special instance graph the number of feasible node delays is finite (but possibly very large). In an implementation one has to consider efficient techniques for handling large sparse matrices [17].

Limits $\tau_{\min}$ and $\tau_{\max}$ are initialized and updated according to [16] using parameter $p_{\text{best}} = 0.00005$. Preliminary tests have shown that these parameter settings work well in general.

5.2 Solution Construction

The method for constructing a solution based on the pheromone values is inspired by the level-based construction heuristic introduced in [18] and runs in time $\mathcal{O}(nB + n^2)$:
(a) For each node a delay value is selected with a probability proportional to the according pheromone value.
(b) All nodes are then sorted by these delay values in ascending order.
(c) The nodes are added in the specified order to the existing tree – initialized with the root node – always choosing the cheapest possible edge without causing a node delay higher than the selected delay. If there is no edge satisfying this constraint, the shortest delay path to the problematic node is added, overriding the given order but guaranteeing a feasible solution.

5.3 Local Improvement

After its construction, a solution is improved either by the VND or by a local search in one of the two neighborhoods Edge-Replace or Component-Renew (see Section 4.1) depending on the instance size. In the latter case the neighborhood Edge-Replace is chosen with probability 0.8 because of its usually higher performance.

5.4 Depositing Pheromones

After each ant constructed and improved a solution the pheromone values are updated. Here the mixed strategy suggested in [16] is used: At the beginning

Table 1. Comparison of GRASP+VND, GVNS and MMAS on random instance sets with 500 and 1000 nodes (B: delay-bound, $\overline{c}$: average final objective values, σ: standard deviations; CPU time limit: 300 seconds; best results are printed bold)

		B	6		20		50		100	
		$\alpha/sr/p$	$\overline{c}$	σ	$\overline{c}$	σ	$\overline{c}$	σ	$\overline{c}$	σ
R500	G+V	0.25	8997.3	672	2048.3	87	942.1	37	616.3	14
	GVNS	dynamic	8703.0	620	1961.5	88	901.1	35	601.1	14
		0.05	8701.1	617	1947.2	88	897.7	35	601.1	15
		0.1	8691.7	618	1938.9	89	893.7	34	599.3	14
		0.15	**8691.6**	618	1942.7	88	894.0	34	599.2	14
		0.2	8696.1	618	1947.8	90	896.4	35	599.8	14
	MMAS	0.6	8727.5	616	1937.4	85	891.4	34	598.0	13
		0.7	8726.4	614	1935.1	85	889.5	34	597.1	12
		0.8	8723.4	612	1932.1	84	**887.5**	34	**596.8**	13
		0.9	8722.4	613	**1930.8**	82	891.2	36	602.2	14
		0.95	8720.4	610	1941.3	83	914.9	40	612.2	14
R1000	G+V	0.25	9775.3	487	2473.0	76	1290.3	31	1026.8	9
	GVNS	dynamic	9497.9	486	2377.7	81	1257.4	33	1020.0	7
		0.05	9397.2	476	2346.9	80	1253.4	31	1020.1	7
		0.1	9412.1	480	2353.6	78	1252.5	31	**1019.4**	7
		0.15	9455.2	487	2365.8	80	1254.7	31	1019.6	7
		0.2	9488.3	485	2374.3	80	1257.0	32	1019.9	7
	MMAS	0.5	9385.3	485	2323.4	75	1243.7	28	1021.3	8
		0.6	9378.4	483	2312.4	73	1239.3	27	1020.9	8
		0.7	9376.4	481	**2308.8**	73	**1238.2**	27	1022.1	10
		0.8	9369.1	478	2309.9	74	1241.3	31	1028.8	14
		0.9	**9367.7**	477	2320.9	76	1281.3	46	1042.8	14

of the search only the best ant of the current iteration is allowed to deposit its pheromones. Later in the search process intensification has higher priority in order to concentrate on the surrounding of the so far best solution. This leads to the following update strategy: The more iterations have been performed, the higher the frequency of reinforcing the pheromone trail of the so far best solution instead of the iteration best one. More precisely, an instance-dependent number of iterations $I = \frac{50000}{|V|}$ is defined. In iterations $[1, I]$ only the iteration best solution deposits pheromones, in $(I, 2I]$ the so far best solution is chosen every fifth iteration, in $(2I, 3I]$ every third iteration, in $(3I, 6I]$ every other iteration and in $(6I, \infty)$ every time. A predefined pheromone decay coefficient p controls the evaporation and enforcement of the pheromone values.

6 Experimental Results

All tests have been executed on a multicore system consisting of Intel Xeon E5540 processors with 2.53 GHz and about 3 GB RAM per core.

Since there are no existing benchmark sets in literature with appropriate graph sizes, we tested our approaches to self-made instance sets called R500 and R1000, now available at `http://www.ads.tuwien.ac.at/~marior/instances/`, each

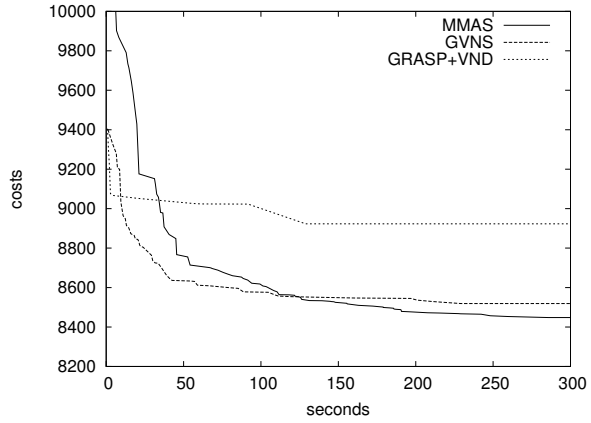

Fig. 1. Typical run characteristics of all three heuristics applied on a R1000 instance with $B = 6$ with a time limit of 300 seconds

containing 30 complete instances with 500 and 1000 nodes, respectively, random integer edge costs and delays uniformly distributed in $[1, 99]$. Depending on the chosen delay-bound B more or less edges can be discarded a priori if $d(e) > B$ (see Section 3.1).

Each result presented in Table 1 is derived from 30 runs with a CPU time limit of 300 seconds for each of the 30 instances. Three metaheuristics are included in the comparison: GRASP+VND from [6], which so far was the best heuristic, and the new GVNS and MMAS (with 5 ants).

All preprocessing steps except the search for alternate constrained paths (see Section 3.2) have been applied before starting the search reducing the complexity of the instances significantly. When having higher time limits the omitted preprocessing step might be advantageous.

The GRASP+VND approach was outperformed by almost all GVNS and MMAS runs almost independent of the parameter settings, which might be explained by the fact that the GRASP+VND has no memory. It "forgets" the solutions of past iterations and therefore cannot build on already obtained information. The MMAS mostly produces the best solutions probably because it has the most effective memory of all three methods in terms of the pheromone values containing the information of many solutions in one data structure.

A matter of high importance when using a MMAS is the fact that it can typically only exhibit its full effectiveness on a rather high number of iterations because of the longer exploration phase in the beginning [16], see Figure 1. When considering the R1000 instances, a full VND improvement of each constructed solution takes much time which leads to a smaller number of iterations within the time limit. The pheromone values therefore have not enough time to converge and the solutions are constructed rather randomly. So a faster local search instead of the VND produces in general worse solutions in a single iteration but yields

higher quality in the end due to the higher number of iterations. For the smaller R500 instances it is better to use the VND improvement since it is fast enough to allow a sufficient number of iterations before time is running out.

When considering strict delay-bounds finding feasible solutions is more difficult and therefore the search gets caught in a local optimum more easily. Rather big changes have to be made to catapult the solution to another basin of attraction. Small changes like replacing a single edge to decrease the costs are often not possible because of a lack of residual delay in the nodes. So choosing the wrong way in the beginning of the search has more impact on the quality of the finally best solution than in cases with looser bounds. This fact can be observed in Table 1 independent of the heuristic method: The stricter the bound the higher the standard deviations.

A too small pheromone decay coefficient in the MMAS causes a fast convergence of the pheromone values and thus disregards diversification; a too high p-value has the opposite effect: the exploration phase lasts too long especially when having a tight time limit.

Relating to the MMAS results the following observation can be made: When tightening the delay-bound the p-value has to be increased to obtain better results. This behavior is another consequence of the facts mentioned above: When having strict bounds the quality and structure of the produced solutions varies much more and by using a higher p-value a single bad solution has not that much influence on the pheromone values because of the smoother evaporation of already deposited pheromones.

Generally speaking, the whole parameter setting of the MMAS heavily depends on the predefined target runtime. Here the small number of five ants speeds up the evolution of the pheromone values which is necessary for the time limit of 300 seconds. Regarding the final results it is not disadvantageous that possibly worse solutions are allowed to update pheromone values.

Applying Wilcoxon signed-rank tests with confidence level 0.95 to the results for $\alpha = 0.25$ (GRASP), $sr = 0.1/0.05$ (GVNS on R500/R1000) and $p = 0.8/0.7$ (MMAS on R500/R1000) yields the following error probabilities P: GVNS and MMAS produce better results than GRASP with $P = 2.2 \cdot 10^{-16}$ for all bounds. MMAS performs better than GVNS on R500 instances with $P = 1$ ($B = 6$), $P = 2.6 \cdot 10^{-13}$ ($B = 20$), $P = 2.2 \cdot 10^{-16}$ ($B = 50, 100$) and on R1000 instances with $P = 7 \cdot 10^{-16}$ ($B = 6$), $P = 2.2 \cdot 10^{-16}$ ($B = 20, 50$), $P = 1$ ($B = 100$).

7 Conclusion

We presented two metaheuristic approaches for solving the rooted delay-constrained minimum spanning tree problem, both outperforming existing approaches. The GVNS benefits from sophisticated neighborhood structures and various shaking moves to quickly converge to high quality solutions, whereas the MMAS with its balanced mix of diversification and intensification built on a fast and diverse solution construction extended with efficient local improvement methods could even exceed the results of the GVNS in most cases.

References

1. Dahl, G., Gouveia, L., Requejo, C.: On formulations and methods for the hop-constrained minimum spanning tree problem. In: Handbook of Optimization in Telecommunications, pp. 493–515. Springer Science + Business Media (2006)
2. Gouveia, L., Paias, A., Sharma, D.: Modeling and Solving the Rooted Distance-Constrained Minimum Spanning Tree Problem. Computers and Operations Research 35(2), 600–613 (2008)
3. Gilbert, E.N., Pollak, H.O.: Steiner minimal trees. SIAM Journal on Applied Mathematics 16(1), 1–29 (1968)
4. Prim, R.C.: Shortest connection networks and some generalizations. Bell System Technical Journal 36, 1389–1401 (1957)
5. Salama, H.F., Reeves, D.S., Viniotis, Y.: The Delay-Constrained Minimum Spanning Tree Problem. In: Blum, C., Roli, A., Sampels, M. (eds.) Proceedings of the 2nd IEEE Symposium on Computers and Communications, pp. 699–703 (1997)
6. Ruthmair, M., Raidl, G.R.: A Kruskal-Based Heuristic for the Rooted Delay-Constrained Minimum Spanning Tree Problem. In: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (eds.) EUROCAST 2009. LNCS, vol. 5717, pp. 713–720. Springer, Heidelberg (2009)
7. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematics Society 7(1), 48–50 (1956)
8. Skorin-Kapov, N., Kos, M.: A GRASP heuristic for the delay-constrained multicast routing problem. Telecommunication Systems 32(1), 55–69 (2006)
9. Xu, Y., Qu, R.: A GRASP approach for the Delay-constrained Multicast routing problem. In: Proceedings of the 4th Multidisplinary International Scheduling Conference (MISTA4), Dublin, Ireland, pp. 93–104 (2009)
10. Ghaboosi, N., Haghighat, A.T.: A Path Relinking Approach for Delay-Constrained Least-Cost Multicast Routing Problem. In: 19th IEEE International Conference on Tools with Artificial Intelligence, pp. 383–390 (2007)
11. Qu, R., Xu, Y., Kendall, G.: A Variable Neighborhood Descent Search Algorithm for Delay-Constrained Least-Cost Multicast Routing. In: Proceedings of Learning and Intelligent OptimizatioN (LION3), pp. 15–29. Springer, Heidelberg (2009)
12. Leggieri, V., Haouari, M., Triki, C.: The Steiner Tree Problem with Delays: A Tight Compact Formulation and Reduction Procedures. Technical report, Universita del Salento, Lecce, Italy (2010)
13. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1(1), 269–271 (1959)
14. Gouveia, L.: Multicommodity flow models for spanning trees with hop constraints. European Journal of Operational Research 95, 178–190 (1996)
15. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. European Journal of Operational Research 130(3), 449–467 (2001)
16. Stützle, T., Hoos, H.: MAX–MIN ant system. Future Generation Computer Systems 16, 889–914 (2000)
17. Duff, I., Erisman, A., Reid, J.: Direct methods for sparse matrices. Oxford University Press, USA (1989)
18. Gruber, M., van Hemert, J., Raidl, G.R.: Neighborhood Searches for the Bounded Diameter Minimum Spanning Tree Problem Embedded in a VNS, EA, and ACO. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, pp. 1187–1194. ACM, New York (2006)