A Multilevel Heuristic for the Rooted Delay-Constrained Minimum Spanning Tree Problem

Martin Berlakovich, Mario Ruthmair, and Günther R. Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
berlmart@a1.net, {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. The problem appears in practice for example when designing a distribution network with a guarantee of timely delivery. Another example is be a centralized broadcasting network where the delaybound represents a quality of service constraint. We introduce a multilevel-based construction heuristic which uses a new measurement for the suitability of edges to create a solution for the problem. In comparison to existing heuristics the main intention is not to create a minimum cost spanning tree, but a solution with a high potential for further improvement. Experimental results indicate that in most cases our approach produces solutions that after local improvement are of higher quality than those of other existing construction techniques.

1 Introduction

Transportation in all its forms is a very important part of our society, be it the transportation of material goods or the transmittal of information. It is usually in the interest of all participants to keep the costs of transportation as low as possible. However, the costs are not always the only deciding factor, expenditure of time is also important. An example would be a shipment organization with a central storage depot providing its customers with goods within a given timeframe, i.e. perishable products. Another example is a central broadcasting service which is required to transmit its information to all receivers within a certain delay boundary. These problems can be classified as network design problems. In order to model this kind of problems the so-called *rooted delay-constrained minimum spanning tree (RDCMST) problem* can be used. The task is to find a spanning tree for a given graph where the edges have cost and delay. No path from a specified root node to any other node may exceed a given delay bound, and the total costs shall be a minimum.

More formally, we are given a graph $G = (V, E)$ with a set V of vertices, a set E of edges, a source vertex $s \in V$ and a delaybound $B > 0$. Additionally

a cost function $c : E \to \mathbb{R}^+$ as well as a delay function $d : E \to \mathbb{R}^+$ assign cost and delay values to the edges, respectively. An optimal solution comprises a spanning tree $T = (V, E'),\ E' \subseteq E$, having minimal costs $c(T) = \sum_{e \in E'} c(e)$ and the delay constraints $\sum_{e \in P(s,v)} d(e) \leq B,\ \forall v \in V$, are satisfied; $P(s, v)$ denotes the unique path between the source s and vertex v. It can be shown that the RDCMST problem is $\mathcal{NP}$-hard by examining a special case, the so-called *hop-constrained minimum spanning tree problem*, where $d(e) = 1,\ \forall e \in E$. This problem is shown to be $\mathcal{NP}$-hard in [1] leading to the conclusion that the more general RDCMST problem is $\mathcal{NP}$-hard, too.

Section **??** gives an overview over existing exact and heuristic approaches for the RDCMST problem. A new measurement of "edge quality" is introduced in Section **??** and our heuristic which is based on this new measurement is introduced in Section **??**. Experimental results are shown and discussed in Section **??**, while Section **??** contains conclusions and some ideas for future work.

2 Previous Work

For the RDCMST problem, exact methods based on integer linear programming have been explored by Leggieri et al. [4] who describe a formulation using lifted Miller-Tucker-Zemlin inequalities. Further approaches have been examined by Gouveia et al. in [3] based on a path formulation solved by column generation, Lagrangian relaxation and a reformulation of the constrained shortest path subproblem on a layered graph. In [7] the latter approach is extended by modeling the whole problem on a layered graph. To overcome the issue of an excessive number of layers in case of a huge set of achievable delay values, a strategy based on iteratively solving smaller layered graphs is presented. However, all these methods can only solve complete graphs with about 100 nodes to proven optimality in reasonable time.

The first heuristic approach, the so-called Prim-based heuristic, was presented in [8]. Here a construction method based on Prim's algorithm to find a minimum spanning tree is described. Starting at the root node the Prim-based heuristic iteratively adds edges adjacent to the existing tree, always choosing the edge with the lowest cost without violating the delay-constraint. The major drawback of this method is that connecting nodes close to the root as cheap as possible can lead to the inability to use cheap edges connecting outlying nodes due to delay-constraints. To counter this problem a more decentralized approach was presented in [5]. The Kruskal-based heuristic (KBH) is, as the name suggests, based on Kruskal's algorithm to find a minimum spanning tree. After sorting the edges by ascending costs all edges are tested whether they can be used to connect components without violating the delay-constraint. In case this first phase results in multiple subtrees a repair algorithm is used to create a final solution. KBH was shown to produce solutions of high quality.

Furthermore, various metaheuristics exist for the problem. In addition to a variable neighborhood descent (VND) in [5], a general variable neighborhood search (VNS) and an ant colony optimization were introduced in [6]. There are

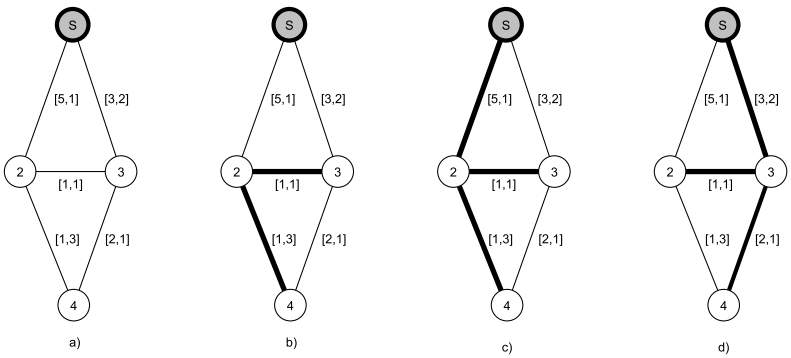

Fig. 1. A short example graph (a) with delaybound B=5. The edge description is read $[costs, delay]$. Adding the cheapest edges to the solution in (b) forces the use of very expensive edges in (c). By also considering the delay a better solution (d) can be created.

many recent publications dedicated to the Steiner tree variant of the RDCMST problem. Here, only a subset of the nodes has to be reached within the given delaybound, the other nodes can optionally be used as intermediate (Steiner) nodes. Several metaheuristics have been applied to this variant, such as GRASP [10], path-relinking [2], and VNS [10]. A hybrid algorithm in [11] combines scatter search with tabu-search, VND, and path-relinking. Furthermore, preprocessing methods are presented in [6] to reduce the size of the input graph significantly in order to speed up the solving process.

3 Ranking Score

In the above construction heuristics the inclusion of an edge with low costs is not necessarily cheap regarding the overall solution. If an edge with low costs but high delay is used it can affect the further construction of the solution negatively. The high delay can force a heuristic to use very expensive edges with low delay in order to not violate the delay constraint. Such decisions sometimes create weak solutions corresponding to poor local optima which even good improvement procedures are not able to overcome. An example is given in Fig. 1.

In an attempt to estimate how promising an edge is, the ranking score is introduced. It is more likely that an edge with comparatively low costs and low delay is part of an optimal solution than an edge with very low costs but high delay. The ranking score

$$score(e) = \left(1 - \frac{r_e^c - 1}{|E|}\right) \cdot \left(1 - \frac{r_e^d - 1}{|E|}\right) \tag{1}$$

describes the relative cost in relation to the delay of an edge $e \in E$ in comparison to other edges; $r_e^c \in \{1, \ldots, |E|\}$ and $r_e^d \in \{1, \ldots, |E|\}$ represent the

cost and delay ranks of edge e obtained by sorting the edges according to costs and delays, respectively. After normalizing the ranking and subtracting from 1 in order to ensure that lower ranks result in higher scores the partial cost and delay scores are multiplied. The resulting ranking score $score(e) \in [0, 1]$ is an indicator for the quality of an edge e.

The ranking score can also be applied on vertices. To calculate the ranking score of a vertex $v \in V$ we sum up the ranking scores of all incident edges. That way the ranking score of a vertex is high if high quality or a high number of edges are connected to that vertex. For example the ranking score of an outlying vertex with few, possibly bad, connections is lower than the ranking score of a central vertex with many connections.

4 Ranking-Based Multilevel Heuristic (RBMH)

The previous construction heuristics referred to in Section **??** are based on adding edges to a partial solution trying to minimize the costs in each step. However, the delay is ignored as long as no constraint violation occurs. This can sometimes lead to relatively poor solutions with a rather low potential for further improvement by local search methods. This motivates a heuristic that uses the above described ranking score to decide which edges should be part of the solution.

Our approach is based on the multilevel paradigm [9], firstly creating a hierarchy of approximations of the original problem by recursive coarsening. After an initial solution has been found on the coarsest level it is iteratively refined in each level obtaining a feasible solution for the original problem in the end. In our case the vertices are iteratively merged to components until only one component is left. The key difference to KBH is the iterative merge process. In each level a number of vertices, including the source vertex, is selected as so-called supervertices. The remaining vertices are connected directly to these supervertices creating multiple subtrees in each level. These subtrees are contracted to vertices in the next level and the process continues until only the source vertex remains. The resulting tree is a spanning tree and due to checks during the merge process it is guaranteed that the delay-constraints are not violated.

4.1 Selecting Supervertices

In each level RBMH has to choose a number of vertices to become supervertices. These supervertices act as root nodes to which the remaining vertices can be connected. For a practical application, i.e. a shipment organization, this can be compared to choosing the site of a regional distribution center and creating a hierarchical network of transportation. The two major questions concerning supervertices are how many vertices should become supervertices and which vertices should be chosen.

The number of supervertices chosen during each level is determined by a user parameter called *superrate*, a simple percentage. A low superrate leads to a low number of supervertices, therefore to a high number of remaining vertices which

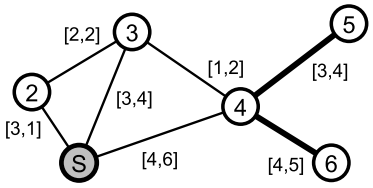

Fig. 2. An example with delaybound $B = 10$. $shortestdelay(3) = 3$, $subtreedelay(4) = 5$
.

have to be connected. The advantage of a low superrate is comparatively fast coarsening since the number of levels will be low, too. However since the number of supervertices is directly related to the number of possible connections for each vertex the search space is smaller. A low superrate is a promising choice if the solution is expected to be a star-like network. Whereas a higher superrate leads to a slower coarsening since more levels can be expected. Note here that the superrate is not directly related to the number of levels due to a mechanism ensuring a feasible solution which will be introduced later. The obvious advantage of a high superrate is that more and maybe better connections are available for each non-supervertex.

The second question is which vertices should become supervertices. Here we apply the ranking score for vertices. The vertices with the highest ranking scores are those with either a high number of connections, thus ensuring a high number of possibilities, or very promising connections. In case of equal ranking scores supervertices are randomly selected making the selection process non-deterministic.

4.2 The Merge Process

After the selection of supervertices the next step is to connect the remaining vertices. Sorted by ascending ranking scores, the only edges considered in the merging process are those between supervertices and other nodes. If

$$shortestdelay(u) + d(\{u, v\}) + subtreedelay(v) \leq B \tag{2}$$

is satisfied for an edge $\{u, v\}$ we know that its use would not violate the delay-constraint. $shortestdelay(v)$ represents the length of the path with the shortest possible delay from the source to vertex v. $subtreedelay(v)$ represents the delay caused by the current subtree of vertex v, see Fig. 2. When checking whether edge $\{3, 4\}$ can be used we have to consider $shortestdelay(3)$ and $subtreedelay(4)$. Summing up these delays plus the edge delay results in an overall delay less than delaybound B. Therefore, this edge can be used to connect vertex 4.

However, there is no guarantee that all non-supervertices can be connected this way. Figure 3 illustrates the problem. For a delaybound of 5 the only possible path to connect vertex 3 is via vertex 4. In case vertex 4 is not a supervertex

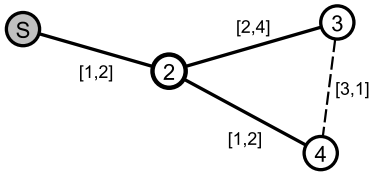

Fig. 3. An example with delaybound $B = 5$. Vertex 3 can only be connected via vertex 4.

Table 1. Comparison of Ranking- and Kruskal-based heuristics without additional improvement, applied on random instance sets with 500 and 1000 nodes (B: delaybound, $\overline{c}$: average final objective value, σ: standard deviation, $t[s]$: average running time in seconds).

	R500						**R1000**					
	RBMH			KBH			RBMH			KBH		
B	$\overline{c}$	σ	$\overline{t[s]}$	$\overline{c}$	σ	$\overline{t[s]}$	$\overline{c}$	σ	$\overline{t[s]}$	$\overline{c}$	σ	$\overline{t[s]}$
10	9282	415	0.35	**7087**	335	0.02	13288	593	1.59	**10296**	484	0.06
30	4817	245	1.11	**3768**	382	0.04	7059	253	5.12	**5064**	460	0.15
50	3711	161	1.94	**2824**	232	0.06	5513	174	8.85	**3243**	360	0.24
75	3142	140	3.00	**2048**	255	0.09	4669	133	13.86	**2185**	232	0.35
100	2812	153	3.99	**1695**	250	0.11	4180	128	19.20	**1605**	196	0.46
150	2802	149	4.62	**1007**	145	0.11	4168	126	19.30	**1165**	131	0.38
200	2802	149	4.44	**784**	124	0.10	4168	126	18.92	**1080**	81	0.35

there is no possibility to connect vertex 3. Therefore, a repair strategy for these problematic vertices is required. If an instance is solvable a feasible path to connect a vertex to the source is given by the shortest-delay-path. For each problem vertex the immediate predecessor in the shortest-delay-path becomes a supervertex in the current level. Additionally, a connection between the new supervertex and a possibly already assigned predecessor is removed. This way a new subtree is created.

After this merge process all non-supervertices are connected and a set of subtrees with supervertices as their root remains. These subtrees are contracted and represent the vertices in the next level, whereas only edges connecting two supervertices are now considered anymore. This process is continued until only the source vertex remains, corresponding to a feasible solution for the original problem. RBMH runs in $\mathcal{O}(|E| \log |E| + |V|^2)$ time.

5 Experimental Results

Our testing environment consists of Intel Xeon E5540 processors with 2.53 GHz and 3 GB RAM per core. The instance sets R500 and R1000 were introduced in [6] and contain 30 complete instances with 500 and 1000 nodes, respectively,

Table 2. Comparison of Ranking- and Kruskal-based heuristics with additional improvement (VND), applied on random instance sets with 500 and 1000 nodes (B: delaybound, $\bar{c}$: average final objective value, σ: standard deviation, $t[s]$: average running time in seconds).

	R500						**R1000**					
	RBMH+VND			KBH+VND			RBMH+VND			KBH+VND		
B	$\bar{c}$	σ	$\overline{t[s]}$	$\bar{c}$	σ	$\overline{t[s]}$	$\bar{c}$	σ	$\overline{t[s]}$	$\bar{c}$	σ	$\overline{t[s]}$
10	4634	225	1.99	**4557**	205	1.45	5290	212	9.33	**5171**	215	7.52
30	**1530**	85	4.42	1554	88	4.37	**1871**	71	23.55	1884	55	20.04
50	**1010**	64	7.99	1042	56	6.22	**1334**	50	33.81	1373	44	32.93
75	**765**	33	10.90	800	37	9.44	**1113**	32	57.75	1146	32	51.42
100	**642**	28	13.64	687	44	12.75	**1038**	12	75.79	1070	32	62.76
150	**547**	11	16.71	587	36	12.25	**1005**	4	74.13	1022	24	57.96
200	**522**	6	13.55	545	27	10.90	**1001**	2	74.58	1008	16	37.65

and random integer edge costs and delays uniformly distributed in $[1, 99]$. Due to RBMH being non-deterministic 30 runs are performed for every instance and average results are used for comparison with KBH presented in [5].

Results without additional improvement show that in general KBH creates much better solutions within shorter runtime. However, RBMH is not directly intended to produce low cost spanning trees but rather use edges which have low costs as well as low delay. Therefore, there may be a lot of improvement potential in a solution provided by RBMH. To use this potential to obtain a solution of higher quality we applied the VND from [5] performing a local search switching between two neighborhood structures based on edge replacement and component renewal, respectively. The results with this additional improvement show that except for very low delaybounds RBMH typically provides a better starting point for further improvement. Especially for very high delay bounds the solutions provided by RBMH can be improved significantly. However, RBMH results also show higher runtimes due to the algorithm's higher complexity and longer improvement phases.

6 Conclusions and Future Work

We introduced a ranking-based multilevel heuristic for the rooted delay-constrained minimum spanning tree problem. By choosing edges with comparably low cost and low delay this construction heuristic produces solutions with a high potential for further improvement. Experimental results indicate that these trees are better starting points for additional local improvement resulting in general in final trees of lower cost compared to solutions generated by the existing Kruskal-based construction heuristic.

In future work we want to extend the approach towards an iterated multilevel approach in which obtained solutions are recoarsened. Furthermore, we intend

to investigate extended variants for the ranking score formula which also use weights to control the influence of costs versus delays to maybe provide even better solutions. Additionally, we want to improve the multilevel heuristic by applying some kind of local search during the refinement phase.

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, chap. 19, pp. 493–515. Springer Science + Business Media (2006)
2. 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)
3. 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)
4. Leggieri, V., Haouari, M., Triki, C.: An Exact Algorithm for the Steiner Tree Problem with Delays. Electronic Notes in Discrete Mathematics 36, 223–230 (2010)
5. 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 (2009)
6. Ruthmair, M., Raidl, G.R.: Variable Neighborhood Search and Ant Colony Optimization for the Rooted Delay-Constrained Minimum Spanning Tree Problem. In: Schaefer, R., et al. (eds.) PPSN XI, Part II. LNCS, vol. 6239, pp. 391–400. Springer (2010)
7. Ruthmair, M., Raidl, G.R.: A Layered Graph Model and an Adaptive Layers Framework to Solve Delay-Constrained Minimum Tree Problems. In: Fifteenth Conference on Integer Programming and Combinatorial Optimization (IPCO XV) (2011), accepted
8. Salama, H.F., Reeves, D.S., Viniotis, Y.: An Efficient Delay-Constrained Minimum Spanning Tree Heuristic. In: Proceedings of the 5th International Conference on Computer Communications and Networks (1996)
9. Walshaw, C.: Multilevel refinement for combinatorial optimisation problems. Annals of Operations Research 131(1), 325–372 (2004)
10. Xu, Y., Qu, R.: A GRASP approach for the Delay-constrained Multicast routing problem. In: Proceedings of the 4th Multidisplinary International Scheduling Conference (MISTA4). pp. 93–104. Dublin, Ireland (2009)
11. Xu, Y., Qu, R.: A hybrid scatter search meta-heuristic for delay-constrained multicast routing problems. Applied Intelligence pp. 1–13 (2010)