Solving the Euclidean Bounded Diameter Minimum Spanning Tree Problem by Clustering-Based (Meta-)Heuristics

Martin Gruber and Günther R. Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
{gruber|raidl}@ads.tuwien.ac.at

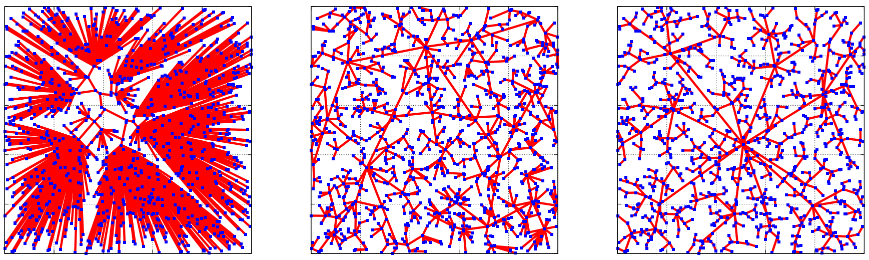
Abstract. The bounded diameter minimum spanning tree problem is an $\mathcal{NP}$-hard combinatorial optimization problem arising in particular in network design. There exist various exact and metaheuristic approaches addressing this problem, whereas fast construction heuristics are primarily based on Prim's minimum spanning tree algorithm and fail to produce reasonable solutions in particular on large Euclidean instances.

In this work we present a method based on hierarchical clustering to guide the construction process of a diameter constrained tree. Solutions obtained are further refined using a greedy randomized adaptive search procedure. Especially on large Euclidean instances with a tight diameter bound the results are excellent. In this case the solution quality can also compete with that of a leading metaheuristic.

1 Introduction

The *bounded diameter minimum spanning tree* (BDMST) problem is a combinatorial optimization problem appearing in applications such as wire-based communication network design when quality of service is of concern, in ad-hoc wireless networks, and also in the areas of data compression and distributed mutual exclusion algorithms.

The goal is to identify a tree-structured network of minimum costs in which the number of links between any pair of nodes is restricted by a constant D, the diameter. More formally, we are given an undirected connected graph $G = (V, E)$ with node set V and edge set E and associated costs $c_e \geq 0$, $\forall e \in E$. We seek a spanning tree $T = (V, E_T)$ with edge set $E_T \subseteq E$ whose diameter does not exceed $D \geq 2$, and whose total costs $c(T) = \sum_{e \in E_T} c_e$ are minimal. This task can also be seen as choosing a *center* – one single node if D is even or an edge in the odd-diameter case – and building a height-restricted tree where the unique path from this center to any node of the tree consists of no more than $H = \lfloor \frac{D}{2} \rfloor$ edges. The BDMST problem is known to be $\mathcal{NP}$-hard for $4 \leq D < |V| - 1$ [1].



(a) CBTC (271.3976). (b) RTC (41.1799). (c) ACO (31.0903).

Fig. 1. A BDMST with $D = 10$ constructed using (a) CBTC, compared to (b) RTC and (c) a solution obtained by an ACO (complete, Euclidean graph with 1000 nodes distributed randomly in the unit square; objective values are given in parentheses).

2 Previous Work

To solve this problem to proven optimality there exist various integer linear programming (ILP) approaches like hop-indexed multi-commodity network flow models [2, 3] or a Branch&Cut formulation based on a more compact model but strengthened by a special class of cutting planes [4]. They all have in common that they are only applicable to relatively small instances, i.e. significantly less than 100 nodes when dealing with complete graphs. For larger instances, metaheuristics have been developed, including evolutionary algorithms [5, 6], a variable neighborhood search, and an ant colony optimization [7] which is currently the leading metaheuristic to obtain high-quality solutions.

In contrast to the large variety of metaheuristic approaches the number of simple and fast construction heuristics applicable to very large instances is limited. They are primarily based on Prim's minimum spanning tree (MST) algorithm and grow a height-restricted tree from a chosen center. One such example is the *center based tree construction* (CBTC) [8]. This approach works reasonably well on instances with random edge costs, but on Euclidean instances this leads to a backbone (the edges near the center) of relatively short edges. The majority of nodes have to be connected to this backbone via rather long edges, see Fig. 1(a). On the contrary, a reasonable solution for this instance, shown in Fig. 1(c), demonstrates that the backbone should consist of a few longer edges to span the whole area to allow the majority of nodes to be connected as leaves by much cheaper edges. In a pure greedy construction heuristic this observation is difficult to realize. In the *randomized tree construction approach* (RTC, Fig. 1(b)) from [8] not the overall cheapest node is added to the partial spanning tree but a random one which then is connected by the cheapest feasible edge. Thus at least the possibility to include longer edges into the backbone at the beginning of the algorithm is increased. For Euclidean instances RTC has been so far the best choice to quickly create a first solution as basis for exact or metaheuristic approaches.

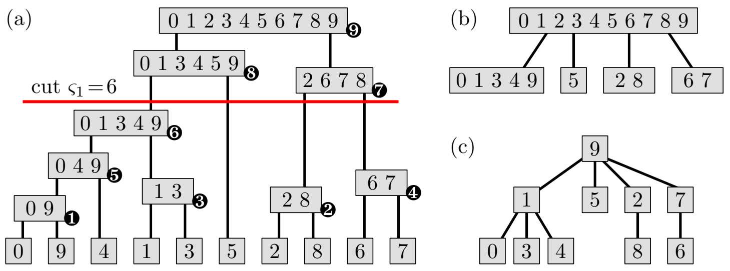


Fig. 2. Hierarchical clustering (a), height-restricted clustering (b), and the resulting BDMST with $D = 4$ (c) after choosing a root for each cluster in (b). In (a) ❶. . . ❾ denote the merge numbers, cf. Section 3.2.

3 Clustering-Based Construction Heuristic

The clustering-based construction heuristic is especially designed for very large Euclidean instances and is based on a hierarchical clustering that guides the algorithm to find a good backbone. It can be divided into three steps: Creating a hierarchical clustering (*dendrogram*) of all instance nodes based on the edge costs, deriving a height-restricted clustering (HRC) from this dendrogram, and finding for each cluster in the HRC a good root (center) node. In the following we will concentrate on the even-diameter case.

3.1 Hierarchical Clustering

For the purpose of creating a good backbone especially for Euclidean instances agglomerative hierarchical clustering seems to provide a good guidance. To get spatially confined areas, two clusters A and B are merged when $\max\{c_{a,b} : a \in A,\ b \in B\}$ is minimal over all pairs of clusters (complete linkage clustering).

The agglomeration starts with each node being an individual cluster, and stops when all nodes are merged within one single cluster. The resulting hierarchical clustering can be illustrated as a binary tree, also referred to as a *dendrogram*, with $|V| - 1$ inner nodes, each representing one merging operation during clustering, and $|V|$ leaves; see Fig. 2(a) for an example with $|V| = 10$.

3.2 Height-Restricted Clustering

After performing the agglomerative hierarchical clustering, the resulting dendrogram has to be transformed into a height-restricted clustering (HRC) for the BDMST, i.e. into a representation of the clustering respecting the diameter and thus the height condition. In general, the dendrogram itself will violate this constraint, see Fig. 2(a). Therefore, some of the nodes in the dendrogram have to be merged to finally get a HRC of height $H - 1$; see Fig. 2(b).

For the quality of the resulting tree this merging of dendrogram nodes is a crucial step. It can be described by $H - 1$ *cuts* through the dendrogram. Preliminary tests revealed that the information at which iteration two clusters have been merged in the agglomeration process, the *merge number*, allows a fine-grained control of the cutting positions (leaves are assigned a merge number of zero). Based on these merge numbers cutting positions ς are computed as

$$\varsigma_i = (|V|-1) - 2^{i \cdot \frac{\log_2 x}{H-1}} \qquad i = 1, \ldots, H-1\,, \tag{1}$$

where x is a strategy parameter that can be interpreted as the number of nodes that shall form the backbone. An experimental evaluation showed that for $D \geq 6$ promising values for x can be found close to $|V|$. Only in case of the smallest possible even diameter of four x should be chosen near $\frac{|V|}{10}$. The approximately best value for x for a specific Euclidean instance and D can be determined by applying binary search for $x \in \left[\frac{|V|}{10}, |V|\right]$. These cutting positions can now be utilized to build the HRC for the BDMST using a simple tree traversal algorithm.

3.3 Determining Root Nodes

Finally, from the height-restricted clustering a BDMST has to be derived by identifying for each (sub-)cluster an appropriate root; cf. Figs. 2(b) and (c). This can be done heuristically in a greedy fashion based on rough cost estimations for each cluster followed by a local improvement step, or by a more sophisticated approach based on dynamic programming.

In the following we will require a more formal and in some points augmented definition of a height-restricted hierarchical clustering. Let $C^0 = \{C^0_1, \ldots, C^0_{|V|}\}$ be the set of clusters at the lowest level 0, where each node of V forms an individual cluster. Moreover, let $C^k = \{C^k_1, \ldots, C^k_{i_k}\}$ be the clustering at the higher levels $k = 1, \ldots, H$. All C^k_i, $i = 1, \ldots, i_k$, are pairwise disjoint, and $C^k_1 \cup C^k_2 \cup \ldots \cup C^k_{i_k} = C^{k-1}$. C^H is the highest level, and it is singleton, i.e. $C^H = \{C^H_1\}$; it refers to all nodes in V aggregated within one cluster. Furthermore, by $V(C^k_i)$ we denote the set of nodes in V represented by the cluster C^k_i, i.e. the nodes part of this cluster and all its sub-clusters at lower levels; $V(C^k) = V(C^k_1) \cup \ldots \cup V(C^k_{i_k}) = V$, and $V(C^k_1) \cap \ldots \cap V(C^k_{i_k}) = \emptyset$, for all $k = 0, \ldots, H$. This definition mainly corresponds to the simple height-restricted clustering previously presented in Fig. 2(b) with the exception that clusters at level zero corresponding to the individual nodes have not been realized explicitly.

Greedy Heuristic with Local Improvement: A simple greedy heuristic to find an initial root for each cluster C^k_i can be based on *stars*, i.e. trees with a diameter of two where a single node v of the cluster acts as center while the remaining nodes $V(C^k_i) \setminus \{v\}$ are connected directly to it. Such a star can be computed for every node $v \in V(C^k_i)$, the center leading to a star of minimal costs for C^k_i is chosen as root for this cluster. The heuristic starts at cluster C^H and assigns roots to clusters top-down until reaching the leaves of the simple height-restricted clustering. Note that a node already selected as root at a level l no longer has to be considered in levels less than l, which can also cause an empty cluster in case all nodes of it are already used as roots at higher levels. This heuristic runs in time $\mathcal{O}(H \cdot |V|^2)$.

In a following local improvement step the selection of root nodes is refined. In case a cluster C_i^k with chosen root v is no leaf of the simple HRC not all nodes of $V(C_i^k) \setminus \{v\}$ will straightly connect to v in the final tree but only the roots of the direct sub-clusters of C_i^k at level $k-1$, cf. Fig. 2(c). This sub-cluster root information was not available in the greedy construction process but now can be used to adapt for each cluster the chosen root node iteratively. This refinement of assigned roots to clusters requires for one iteration time $\mathcal{O}(H \cdot \delta^{\max} \cdot |V|)$, where $\delta^{\max}$ is the maximal branching factor in the HRC.

Dynamic Programming: There are multiple effects on the tree when choosing a specific node v as root for a cluster, e.g. v no longer can act as root for one of the sub-clusters, but it also has not be connected as leaf to the tree. These effects increase the complexity of deriving an optimal BDMST for a given hierarchical clustering to such an extent that it is in general computationally unattractive. Nevertheless, when making certain assumptions it is possible to formulate an efficient dynamic programming approach for this problem.

Let $c(C_i^k, v)$ denote the minimum costs of the subtree of the BDMST defined by the cluster C_i^k if it is rooted at node $v \in V(C_i^k)$, i.e. node v has been chosen as root for cluster C_i^k. Beside other implications one major point when choosing a node v as root is that it no longer has to be connected elsewhere in the tree. When computing $c(C_i^k, v)$ and selecting another node w from the same sub-cluster $C_{j'}^{k-1}$ that v is also part of, then the costs $c(C_{j'}^{k-1}, w)$ also contains the costs to connect (perhaps as root of one of the sub-clusters, more likely as a leaf of the BDMST) node v. To exactly compute the contribution of v to the costs of $c(C_{j'}^{k-1}, w)$ is in practice usually not worth the (huge) effort, in particular when considering the costs of edges between root nodes in relation to the costs of connecting a leaf to the tree via a short edge, which is the goal of the whole clustering heuristic.

This observation can be used to formulate an approximate dynamic programming approach utilizing a correction value κ_v for each node $v \in V$ which estimates the costs arising when v has to be connected as leaf to the BDMST. There are various possibilities to define these correction values, preliminary tests showed that a simple choice usually is sufficient: For each cluster at level one the cheapest star is determined, and for a node v of such a cluster, κ_v are the costs to connect it to the center of the best star. The costs $c(C_i^k, v)$ can now be recursively defined for each level and node of a cluster as follows:

$$c(C^0_{\mathrm{ord}(v)}, v) = 0 \qquad \forall v \in V \tag{2}$$

$$\phi(C_i^k, v) = \sum_{C_j^{k-1} \in C_i^k \setminus \{C_{j'}^{k-1}\}} \min_{u \in V(C_j^{k-1})} \Big(c_{v,u} + c(C_j^{k-1}, u) \Big)$$

$$c(C_i^k, v) = \min \Big(c(C_{j'}^{k-1}, v),\ c_{v,w} + c(C_{j'}^{k-1}, w) - \kappa_v \Big) + \phi(C_i^k, v)$$

$$\forall k = 1, \ldots, H;\ \forall v \in V(C_i^k);\ C_{j'}^{k-1} \in C_i^k \mid v \in V(C_{j'}^{k-1});\ w \in V(C_{j'}^{k-1}) \mid w \neq v \tag{3}$$

Algorithm 1: refineCuts(ς)

```
input  : cutting positions $\varsigma_i$, $i = 1, \ldots, H-1$
output: improved cutting positions
1  $T^* \leftarrow$ buildTree($\varsigma$) ;                                  // currently best BDMST $T^*$
2  $\varsigma^* \leftarrow \varsigma$ ;                                      // currently best cutting positions $\varsigma^*$
3  clear cache for sets of cutting positions and insert $\varsigma$;
4  $lwi \leftarrow 0$ ;                                                     // loops without improvement
5  repeat
6      for $i = 1, \ldots, H-1$ do
7          if $i = 1$ then $\Delta \leftarrow (|V|-1) - \varsigma^*_1$ else $\Delta \leftarrow \varsigma^*_{i-1} - \varsigma^*_i$;
8          repeat $\varsigma_i \leftarrow \lfloor \varsigma^*_i + \Delta \cdot N(\mu, \sigma^2) + 0.5 \rfloor$; until $check(\varsigma_i)$ is ok
9      if $\varsigma \in$ cache then $lwi \leftarrow lwi + 1$ and continue;
10     insert $\varsigma$ into cache;
11     $T \leftarrow$ buildTree($\varsigma$);
12     if $c(T) < c(T^*)$ then $T^* \leftarrow T$; $\varsigma^* \leftarrow \varsigma$; $lwi \leftarrow 0$; else $lwi \leftarrow lwi + 1$;
13 until $lwi \geq l_{\max}$ ;
```

At level zero each node is a single cluster. Therefore, in (2) the costs of the corresponding subtrees can be initialized with zero (ord(v) assigns each node $v \in V$ an unique index within 1 and $|V|$). Then the costs $c(C_i^k, v)$ are composed of two parts: The minimal costs of either using directly the subtree rooted at v from level $k-1$ or another node w from the same sub-cluster $C_{j'}^{k-1}$, plus for all remaining direct sub-clusters the minimal costs to connect a node u of a sub-cluster with its subtree to v, referred to as $\phi(C_i^k, v)$ in (3). After deriving all these costs in a bottom-up fashion, optimal root nodes leading to these costs can be chosen top-down in a second pass. This dynamic programming approach computes roots for clusters within time $\mathcal{O}(H \cdot |V|^2)$.

Connecting Leaf Nodes: When strictly following the clustering the leaves of the BDMST has to connect to the root nodes of their respective clusters. However, this strategy neglects the fact that there are in general much cheaper opportunities since a leaf node can be attached to any root node of a cluster without violating the height and therefore the diameter restriction. Thus, releasing the leaves from their strict membership to a specific cluster and to allow them to establish the cheapest possible connection to an arbitrary root can improve the solution quality substantially.

4 Refining Cutting Positions

In Section 3.2 the computation of initial cutting positions ς_i, $i = 1, \ldots, H-1$, through the dendrogram to receive a height-restricted clustering has been presented. Since these ς_i have a formidable impact on solution quality we additionally implemented an approach similar to a greedy randomized adaptive search procedure (GRASP) to further refine them, see Algorithm 1. In each iteration all cutting positions of the currently best solution are perturbated using the difference Δ to the next lower indexed cutting position (for ς_1 the value $(|V|-1) - \varsigma_1$ is used), multiplied with a Gaussian distributed random value $N(\mu, \sigma^2)$.

Table 1. Averaged objective values over all 15 Euclidean Steiner tree instances from [9] with 1000 nodes for various even diameter bounds and (meta-) heuristics, the standard deviations are given parentheses.

	without VND				with VND			
D	CBTC	RTC	CL	t(CL) [s]	RTC	CL	ACO	t(CL) [s]
4	329.026 (6.02)	146.492 (3.88)	**68.323** (0.70)	2.54 (0.09)	65.206 (0.55)	**65.160** (0.56)	65.801 (0.48)	5.56 (1.01)
6	306.266 (9.02)	80.864 (2.40)	**47.170** (4.61)	4.55 (0.49)	41.458 (0.36)	**41.313** (0.50)	42.117 (0.29)	9.94 (1.52)
8	288.384 (7.52)	53.253 (1.33)	**36.941** (1.34)	5.92 (0.42)	35.051 (0.35)	**34.217** (0.29)	34.749 (0.21)	11.61 (1.61)
10	266.366 (9.01)	41.120 (0.68)	**33.341** (0.66)	6.79 (0.42)	32.118 (0.31)	**30.970** (0.24)	31.039 (0.22)	13.43 (2.16)
12	250.002 (8.01)	35.759 (0.47)	**31.956** (0.44)	7.11 (0.33)	30.290 (0.29)	29.180 (0.26)	**28.636** (0.22)	14.68 (2.49)
14	237.140 (6.28)	33.364 (0.30)	**31.018** (0.33)	7.00 (0.64)	29.094 (0.28)	28.009 (0.23)	**26.652** (0.30)	15.05 (3.00)
16	224.312 (5.72)	32.196 (0.24)	**30.429** (0.29)	7.20 (0.72)	28.243 (0.28)	27.136 (0.19)	**25.576** (0.15)	15.63 (2.89)
18	210.987 (7.63)	31.583 (0.24)	**30.135** (0.27)	7.32 (0.81)	27.601 (0.27)	26.560 (0.20)	**24.881** (0.18)	16.78 (3.61)
20	197.177 (7.99)	31.268 (0.22)	**30.038** (0.28)	7.57 (0.76)	27.109 (0.26)	26.108 (0.23)	**24.370** (0.14)	18.54 (3.89)
22	183.016 (8.03)	31.086 (0.22)	**30.074** (0.28)	8.56 (0.98)	26.698 (0.28)	25.805 (0.21)	**24.013** (0.16)	21.39 (5.19)
24	172.825 (10.59)	30.992 (0.23)	**30.160** (0.27)	8.28 (1.41)	26.365 (0.27)	25.452 (0.24)	**23.772** (0.19)	21.36 (6.42)

To derive an actual BDMST from the cutting positions ς in `buildTree(`ς`)` a fast construction heuristic should be applied like the greedy heuristic with local search presented in the previous Section 3.3. To avoid redundant computations a cache is used to identify sets of cutting positions ς already evaluated. Furthermore, a new cutting position ς_i is only accepted if it lies within the interval $[|V|-2, 1]$ and if it differs from all ς_j, $j < i$, which is tested in `check(`ς_i`)`. The whole refinement process is stopped when $l_{\max}$ iterations without improvement have been performed, or no sets of new cutting positions could be found.

5 Computational Results

The experiments have been performed on an AMD Opteron 2214 (2.2GHz) utilizing benchmark instances already used in the corresponding literature from Beasley's OR-Library [9] originally proposed for the Euclidean Steiner tree problem. These complete instances contain point coordinates in the unit square, and the Euclidean distances between each pair of points are taken as edge costs. As performance differences are more significant on larger instances, we restrict our attention here to the 15 largest instances with 1000 nodes.

Table 1 summarizes the results obtained for various heuristics. Given are the objective values averaged over all 15 instances (30 independent runs per instance), together with the standard deviations in parentheses. Considered are the two established construction heuristics CBTC and RTC as well as the clustering heuristic CL. In GRASP a mean μ of 0 and, after preliminary tests, a variance σ^2 of 0.25 was used, and the procedure was aborted after $l_{\max} = 100$ iterations without improvement. The time (in seconds) listed is the over all instances averaged running time of the clustering heuristic which was also used as time limit for the corresponding executions of CBTC and RTC. To verify statistical significance paired Wilcoxon signed rank tests have been performed.

Clearly, CBTC is not suited for this type of instances, its strength lies in problems with random edge costs. CL outperforms RTC for every diameter bound significantly, where the gap in solution quality is huge when D is small. When applying a strong variable neighborhood descend (VND) as proposed in [7] to the best solutions of the various construction heuristics the differences are flattened. Nevertheless, the BDMSTs derived from CL solutions are of higher quality in general. On instances with a small diameter bound these trees – computed in a few seconds – can also compete with results from the leading metaheuristic, the ACO from [7], obtained after one hour of computation.

6 Conclusions

On the more difficult to solve Euclidean BDMST instances fast construction heuristics proposed so far were primarily based on Prim's MST algorithm and were too greedy to compute reasonable results. We presented a constructive heuristic that exploits a hierarchical clustering to guide the process of building a diameter-constrained tree.

In particular on large Euclidean instances the BDMSTs obtained by the clustering heuristic are in general of high quality and outperform the other construction heuristics significantly, especially when the diameter bound is tight. When using a strong VND to further improve these solutions they can also compete with results from an ACO, currently the leading metaheuristic for this problem.

References

1. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
2. Gouveia, L., Magnanti, T.L.: Network flow models for designing diameter-constrained minimum spanning and Steiner trees. Networks **41**(3) (2003) 159–173
3. Gouveia, L., Magnanti, T.L., Requejo, C.: A 2-path approach for odd-diameter-constrained minimum spanning and Steiner trees. Networks **44**(4) (2004) 254–265
4. Gruber, M., Raidl, G.R.: (Meta-)heuristic separation of jump cuts in a branch&cut approach for the bounded diameter minimum spanning tree problem (2008) submitted to a special issue on Matheuristics of Operations Research/Computer Science Interface Series, Springer.
5. Raidl, G.R., Julstrom, B.A.: Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In Lamont, G., et al., eds.: Proc. of the ACM Symposium on Applied Computing, ACM Press (2003) 747–752
6. Singh, A., Gupta, A.K.: Improved heuristics for the bounded-diameter minimum spanning tree problem. Soft Computing – A Fusion of Foundations, Methodologies and Applications **11**(10) (2007) 911–921
7. 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 Keijzer, M., et al., eds.: Proc. of the Genetic and Evolutionary Computation Conference 2006. Volume 2. (2006) 1187–1194
8. Julstrom, B.A.: Greedy heuristics for the bounded diameter minimum spanning tree problem. Journal of Experimental Algorithmics (JEA) **14** (2009) 1.1:1–1.1:14
9. Beasley, J.: OR-Library: Capacitated MST (2005) `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/capmstinfo.html`.