Exploiting Hierarchical Clustering for Finding Bounded Diameter Minimum Spanning Trees on Euclidean Instances

Martin Gruber
gruber@ads.tuwien.ac.at

Günther R. Raidl
raidl@ads.tuwien.ac.at

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9–11/186-1, 1040 Vienna, Austria

ABSTRACT

The bounded diameter minimum spanning tree problem is an NP-hard combinatorial optimization problem arising, for example, in network design when quality of service is of concern. 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.

A method based on hierarchical clustering to guide the construction process of a diameter constrained tree is presented. Solutions obtained are further refined using a greedy randomized adaptive search procedure. Based on the idea of clustering we also designed a new neighborhood search for this problem. 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, whereas the computation only needs a fraction of the time.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; G.1.6 [**Numerical Analysis**]: Optimization—*Constrained optimization*

General Terms

Algorithms

Keywords

Bounded Diameter Minimum Spanning Tree, Construction Heuristics, Greedy Randomized Search, Dynamic Programming, Local Improvement

1. INTRODUCTION

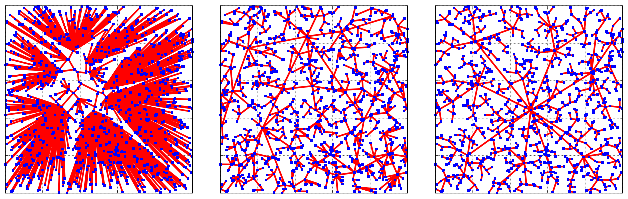
The *bounded diameter minimum spanning tree* (BDMST) problem is a combinatorial optimization problem appearing in applications such as wire-based communication network design when quality of service is of concern, in ad-hoc wireless networks [1], and also in the areas of data compression and distributed mutual exclusion algorithms [16, 5].

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 $4 \leq D < |V| - 1$ [6].

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 [7, 8] or a Branch&Cut formulation based on a more compact model but strengthened by a special class of cutting planes [9]. They all have in common that they are only applicable to relatively small instances, i.e. significantly fewer than 100 nodes when dealing with complete graphs. For larger instances, metaheuristics have been developed, including evolutionary algorithms [15, 18], a variable neighborhood search, and an ant colony optimization [11] 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 that can also be applied to very large instances is limited. They are primarily based on Prim's minimum spanning tree (MST) algorithm [14] and grow a height-restricted tree from a chosen center. One such example is the *center based tree construction* (CBTC) [13]. 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)



(a) CBTC (271.4). (b) RTC (41.18). (c) ACO (31.09).

Figure 1: A diameter constrained tree with $D = 10$ constructed using (a) the CBTC heuristic, compared to (b) RTC (best solution from 1000 runs) and (c) a solution obtained by an ant colony optimization approach (complete, Euclidean graph with 1000 nodes distributed randomly in the unit square, the corresponding objective values are given in parentheses).

of relatively short edges and the majority of the nodes have to be connected to this backbone via rather long edges, see the example in 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 large number of remaining 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 [13] not the cheapest of all nodes is always added to the partial spanning tree but the next node is chosen at random and then 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.

In the following we will introduce a new construction heuristic for the BDMST problem which is especially suited for very large Euclidean instances. It is based on a hierarchical clustering that guides the algorithm to find a good backbone. This approach is then refined by a local improvement method and extended towards a greedy randomized adaptive search procedure (GRASP) [17]. A preliminary version of this work can be found in [10].

3. THE CLUSTERING HEURISTIC

The clustering-based construction heuristic 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.

3.1 Hierarchical Clustering

For the purpose of creating a good backbone especially for an Euclidean instance of the BDMST problem 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 [12]).

The agglomeration starts with each node being an individual cluster, and stops when all nodes are merged within one

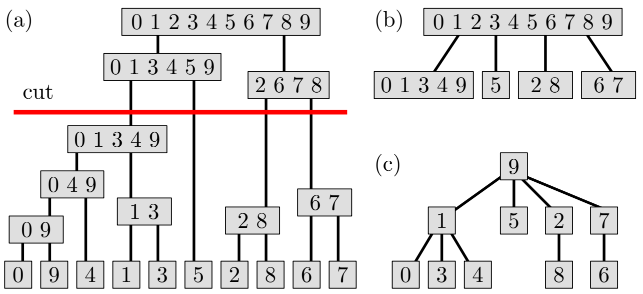


Figure 2: Hierarchical clustering (a), height-restricted clustering (b), and the resulting diameter constrained tree with $D = 4$ (c) after choosing a root for each cluster in (b).

single cluster. The resulting hierarchical clustering can be illustrated as a binary tree, also referred to as a *dendrogram*, with $|V|$ leaves and $|V| - 1$ inner nodes each representing one merging operation during clustering; see Fig. 2(a) for an example with $|V| = 10$. An inner node's distance from the leaves indicates when the two corresponding clusters – relative to each other – have been merged.

3.2 Height-Restricted Clustering

After performing the agglomerative hierarchical clustering, the resulting dendrogram is 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. The dendrogram itself cannot directly act as HRC since in general it will violate this constraint, see Fig. 2(a). Therefore, some of the nodes in the dendrogram have to be merged to finally get a tree of height $H - 1$, the HRC for the BDMST; see Fig. 2(b) for a diameter of $D = 4$.

For the quality of the resulting tree this merging of dendrogram nodes is a crucial step, worth significant effort. It can be described by $H - 1$ *cuts* through the dendrogram defining which nodes of it will also become part of the height-restricted clustering and which are merged with their parent clusters. As an example, starting at the root containing all instance nodes agglomerated within one single cluster the cut illustrated in Fig. 2(a) defines the dendrogram nodes $\{0, 1, 3, 4, 9\}$, $\{5\}$, $\{2, 8\}$, and $\{6, 7\}$ to become direct successors of the root cluster in the height-restricted clustering.

One fundamental question arising in this context is the way of defining the cutting positions through the dendrogram. After preliminary tests, the identification of the precise iteration at which two clusters have been merged in the agglomeration process turns out to be a good criterion. This *merge number* (or *merge#*), which allows a fine-grained control of the cutting positions, can be stored within each node of the dendrogram, with the leaves having a merge number of zero and the root $|V| - 1$.

Based on the merge numbers cutting positions ς are computed as

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

where x is a strategy parameter. This formula is motivated by a perfectly balanced tree, where parameter x can be interpreted as the number of nodes that shall form the backbone.

These cutting positions can now be used to build the height-restricted clustering for the BDMST, as depicted in

Algorithm 1: buildHRC(p_d, p_p, ς, j)

```
input  : reference to dendrogram node $p_d$; reference to
         parent cluster in the HRC $p_p$; cutting positions
         $\varsigma_i$, $i = 1, \ldots, H-1$; current cut index $j$
output: height-restricted clustering for the BDMST

if $p_d.merge\# > \varsigma_j$ then
    forall children $p_c$ of $p_d$ do
        buildHRC($p_c$, $p_p$, $\varsigma$, $j$);
else
    create new HRC node $p_n$ for instance nodes
    agglomerated in $p_d$;
    connect $p_n$ to its parent cluster $p_p$ within the HRC;
    if $j < H-1$ then
        forall children $p_c$ of $p_d$ do
            buildHRC($p_c$, $p_n$, $\varsigma$, $j+1$);
```

Algorithm 1. The recursion is started with the root of the dendrogram (p_d), a reference to the newly created root node of the HRC (p_p), the computed cutting positions (ς), and 1 for the current cut index j.

An experimental evaluation with a simple greedy construction heuristic described in the next section revealed that for $D \geq 6$ promising values for x can be found close to $|V|$; see Fig. 3. Only in case of the smallest possible even diameter of four the picture is inverted and x should be chosen near $\frac{|V|}{10}$. The rather continuous and mostly monotonic increase or decrease of the curve further suggests to apply binary search to determine an approximately best value for x for a specific Euclidean instance and diameter bound, if time allows and the heuristic can be run multiple times.

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 more sophisticated approaches based on dynamic programming [4].

In the following we will require a more formal and in some points augmented definition of a height-restricted hierarchical clustering. Let $C^0 = \{C_1^0, \ldots, C_{|V|}^0\}$ be the set of clusters at the lowest level 0, where each node of V forms an individual cluster. Moreover, let $C^k = \{C_1^k, \ldots, C_{i_k}^k\}$ be the clustering at the higher levels $k = 1, \ldots, H$. All C_i^k, $i = 1, \ldots, i_k$, are pairwise disjoint, and $C_1^k \cup C_2^k \cup \ldots \cup C_{i_k}^k = C^{k-1}$. C^H is the highest level, and it is singleton, i.e. $C^H = \{C_1^H\}$; it refers to all nodes in V aggregated within one cluster. Furthermore, by $V(C_i^k)$ we denote the set of nodes in V represented by the cluster C_i^k, i.e. the nodes part of this cluster and all its sub-clusters at lower levels; $V(C^k) = V(C_1^k) \cup \ldots \cup V(C_{i_k}^k) = V$, and $V(C_1^k) \cap \ldots \cap V(C_{i_k}^k) = \emptyset$, for all $k = 0, \ldots, H$.

This definition mainly corresponds to the simple height-restricted clustering previously presented in Fig. 2(b) and computed by Algorithm 1, with two exceptions: The clusters at level zero corresponding to the individual nodes have not been realized explicitly, and not all leaves of the HRC created by Algorithm 1 have to be found at level one. In the latter case such a leaf can only contain exactly one node $v \in V$, therefore the HRC can be augmented with (virtual) nodes to connect the corresponding cluster at level zero with a leaf at a level ≥ 2.

Algorithm 2: greedyRoots(r)

```
input  : root $r$ of the HRC
output: a root node for each cluster, and if $D$ is odd a
        center edge for the root cluster of the HRC

forall $v \in V$ do $available[v] \leftarrow$ true;

if $D$ is even then
    assignRoot($r$);
else
    forall children $r_c$ of $r$ do
        $r_c.stars \leftarrow ()$;
        foreach node $v \in V(r_c)$ do
            compute star $s_v$: connect to $v$ all nodes
            $V(r_c) \setminus \{v\}$ of cluster $r_c$;
            $r_c.stars$.append($s_v$);
        sort $r_c.stars$ ascending according to the costs of
        the stars;
        $r_c.root \leftarrow v$ of least cost star $s_v \in r_c.stars$;

    $r.root \leftarrow$ best center edge for the roots of the child
    clusters $r_c$;
    for both center nodes $c_1$ and $c_2$ do
        $available[c_i] \leftarrow$ false;

    forall children $r_c$ of $r$ do
        if not $available[r_c.root]$ then
            $r_c.root \leftarrow$ next best root node based on
            $r_c.stars$;
        $available[r_c.root] \leftarrow$ false;
        forall children $p_c$ of $r_c$ do assignRoot($p_c$);
```

Algorithm 3: assignRoot(p)

```
input  : reference $p$ to a node of the HRC
output: for cluster $p$ a sorted list $p.stars$ of diameter 2
        trees and a root $p.root$

$p.root \leftarrow \emptyset$;
$p.stars \leftarrow ()$;

foreach node $v \in V(p)$ do
    compute star $s_v$: connect to $v$ all nodes $V(p) \setminus \{v\}$
    of current cluster $p$;
    $p.stars$.append($s_v$);
sort $p.stars$ ascending according to costs of the stars;

scan $p.stars$ from beginning for first $s_v$ where
$available[v] =$ true;
if such a star $s_v$ could be found then
    $p.root \leftarrow v$;
    $available[v] \leftarrow$ false;
    forall children $p_c$ of $p$ do assignRoot($p_c$);
```

3.3.1 Greedy Heuristic with Local Improvement

A simple greedy heuristic to find an initial root for each cluster C_i^k can be based on so-called *stars*, i.e. trees with a diameter of two where a single node v of the cluster acts as center while the remaining nodes $V(C_i^k) \setminus \{v\}$ are con-

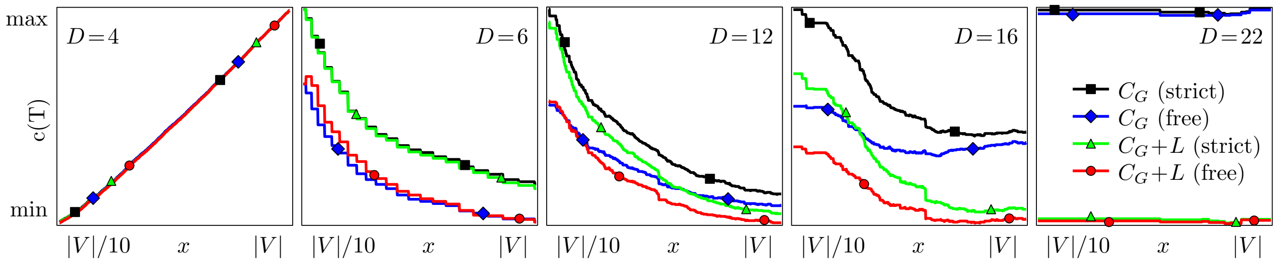


Figure 3: Obtained objective values (scaled to minimal and maximal values) over parameter x of Equation (1) (ranging from $\frac{|V|}{10}$ to $|V|$) for various diameter bounds on an Euclidean instance with 1000 nodes distributed randomly in the unit square. The trees were computed using the simple greedy construction heuristic C_G (Section 3.3) with and without local improvement (L), respectively with free leaf nodes or leaves strictly following the clustering (see Section 3.4 for details).

nected directly to it. Such a star can be computed for every node belonging to the cluster, the center node v leading to a star of minimal costs for C_i^k 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.

Algorithms 2 and 3 illustrate this heuristic in more detail. An array *available* is used to indicate whether a node still can be selected as root in a (sub-)cluster. The even-diameter case is much simpler to handle: For each cluster represented by a node in the simple HRC all possible stars are computed and gathered within a list. This list is sorted in ascending order according to the costs of the stars, and the still available node leading to the cheapest star is chosen as root for the current cluster.

While this procedure can also be used for the root cluster C_1^H of the height-restricted clustering when D is even, not a single node but an edge has to be selected as center of the BDMST in case the diameter bound is odd. The corresponding algorithm would be to compute the cheapest BDMST with a diameter of three for the whole instance and use the resulting center edge also as center for the root cluster. However, in general this would lead to a much too long center edge because in a good BDMST this edge becomes shorter with increasing diameter bound since it no longer has to span a larger area. To make a better choice in a first step a reasonable root node is computed for every cluster C_i^{H-1} at level $H-1$. These roots are the only nodes that have to be linked directly to the center edge. Based on this observation now a more suitable center edge can be determined by computing a diameter three BDMST only considering the connection costs of the root nodes of the clusters C_i^{H-1}.

While this heuristic runs in time $\mathcal{O}(H \cdot |V|^2)$ when D is even, the selection of the center edge in the odd-diameter case adds a term of $\mathcal{O}(\delta^r \cdot |E|)$, with δ^r being the branching factor of the root cluster in the HRC, leading to an overall runtime complexity of $\mathcal{O}(\delta^r \cdot |E| + H \cdot |V|^2)$.

In a following local improvement step the selection of root nodes (and the center edge) is refined. In case a cluster C_i^k with chosen root v is no leaf of the simple height-restricted clustering not all nodes of $V(C_i^k)\backslash\{v\}$ 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

Algorithm 4: locallyImprove(r)

```
input  : root $r$ of the HRC
output: locally improved roots for each cluster of the
        HRC
$costs^* \leftarrow$ costs $c(T)$ of the current BDMST $T$ derived
from the HRC;
$T^* \leftarrow T$;
repeat
    $improved \leftarrow$ false;
    forall $v \in V$ do $available[v] \leftarrow$ true;
    $r.root \leftarrow$ best center to connect current roots of all
    child clusters $r_c$ of $r$;
    update $available[\cdot]$ accordingly;
    recursively find for each sub-cluster of $r$ the
    currently local optimal root:
        • consider only nodes $v_i$ with
          $available[v_i]$ = true,
        • consider connection costs to root of parent
          cluster,
        • if no leaf of the HRC: consider costs to
          connect the current roots of direct successor
          clusters,
        • always update $available[\cdot]$ accordingly;
    evaluate current BDMST $T$ derived from the
    assigned roots in the HRC;
    if $c(T) < costs^*$ then
        $T^* \leftarrow T$;
        $costs^* \leftarrow c(T)$;
        $improved \leftarrow$ true;
until $improved = false$;
restore best tree $T^*$;
```

root information was not available in the greedy construction process since the assignment from root nodes to clusters was performed top-down but now can be used to adapt for each cluster the chosen root node iteratively, see Algorithm 4. This refinement of assigned roots to clusters requires for one iteration time $\mathcal{O}(H \cdot \delta^{\max} \cdot |V|)$ if D is even, where $\delta^{\max}$ is the maximal branching factor in the height-restricted clustering, and $\mathcal{O}(\delta^r \cdot |E| + H \cdot \delta^{\max} \cdot |V|)$ in the odd-diameter case.

Attention has to be payed to the fact that a local improving move (new root for a specific cluster C_i^k) not necessarily leads to an improvement of the overall BDMST. Choosing a node u instead of v as root node for C_i^k can have various effects on this part of the tree. E.g. u no longer can act as root for one of the clusters at a lower level; moreover, v now has to be connected as a new leaf to the BDMST if not chosen as a root within one of the sub-clusters of C_i^k. As a consequence, the stopping criterion is not based on the existence or absence of an local improvement move but on the costs of the whole derived BDMST.

3.3.2 *Dynamic Programming*

The multiple effects on the tree when choosing a specific node as root for a cluster 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 restrictions on the choice of root nodes, 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. These costs can now be recursively defined for each level and node of a cluster as follows:

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

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

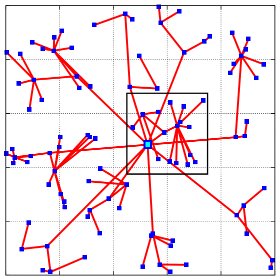
$$c(C_i^k, v) = c(C_{j'}^{k-1}, v) + \phi(C_i^k, v)$$

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

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 $|V|$). In the remaining levels we restrict the root for a cluster C_i^k to nodes that are already roots in one of its direct sub-clusters $C_j^{k-1} \in C_i^k$. Then the costs $c(C_i^k, v)$ are composed of the costs of the subtree rooted at v at level $k-1$ 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.

Limiting the potential roots of a cluster to root nodes within one of its sub-clusters obviously leads to suboptimal trees, especially when the diameter bound is loose and each root node only has very few connections. Moreover, using the whole subtree rooted at v from a cluster at level $k-1$ for a cluster at level k implies that this subtree is moved one edge towards the center of the BDMST and therefore does not exploit the full possible height, a problem arising for every cluster at every level $k \geq 2$.

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.


(a) BDMST with leaves following the clustering.

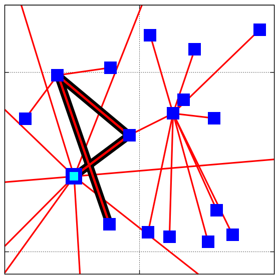

(b) Problematic path in the solution (a).

Figure 4: Problem arising when strictly following the clustering on a complete Euclidean instance with 100 nodes and $D = 6$. In (b) the interesting area of (a) near the center of the BDMST is shown enlarged, a problematic path is highlighted.

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: The subtrees computed at level one correspond to stars with diameter two. 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. This now leads to the following reformulation of the recursion to compute the costs $c(C_i^k, v)$:

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

$$\forall k = 1, \ldots, H, \quad \forall v \in V(C_i^k), \quad C_{j'}^{k-1} \in C_i^k \mid v \in V(C_{j'}^{k-1}),$$
$$w \in V(C_{j'}^{k-1}) \mid w \neq v \tag{4}$$

Both dynamic programming approaches compute roots for clusters within time $\mathcal{O}(H \cdot |V|^2)$ and $\mathcal{O}(|V| \cdot |E| + H \cdot |V|^2)$ for the even- and odd-diameter cases, respectively.

3.4 Inherent Problem of Clustering

One problem arising when strictly following the clustering to the leaves of the BDMST – in particular when the diameter bound is weak in comparison to the number of nodes – is illustrated in Fig. 4. This situation can be observed when a node of a (sub-)cluster is chosen to be part of the backbone (a root node), and other nodes close to it not. The node

Algorithm 5: refineCuts(ς)

```
input  : cutting positions $\varsigma_i$, $i = 1, \ldots, H-1$
output: improved cutting positions
$T^* \leftarrow$ buildTree($\varsigma$) ;      // currently best BDMST $T^*$
$\varsigma^* \leftarrow \varsigma$ ;    // currently best cutting positions $\varsigma^*$
clear cache for sets of cutting positions and insert $\varsigma$;
$lwi \leftarrow 0$ ;                // loops without improvement
repeat
    for $i = 1, \ldots, H-1$ do
        if $i = 1$ then $\Delta \leftarrow (|V|-1) - \varsigma_1^*$ else
        $\Delta \leftarrow \varsigma_{i-1}^* - \varsigma_i^*$;
        repeat
            $\varsigma_i \leftarrow \lfloor \varsigma_i^* + \Delta \cdot N(\mu, \sigma^2) + 0.5 \rfloor$;
        until check($\varsigma_i$) is ok;
    if $\varsigma \in$ cache then
        $lwi \leftarrow lwi + 1$;
        continue;
    insert $\varsigma$ into cache;
    $T \leftarrow$ buildTree($\varsigma$);
    if $c(T) < c(T^*)$ then
        $T^* \leftarrow T$;
        $\varsigma^* \leftarrow \varsigma$;
        $lwi \leftarrow 0$;
    else
        $lwi \leftarrow lwi + 1$;
until $lwi \geq l_{\max}$ ;
```

right below the center of the BDMST in Fig. 4(a) and the close-up (b) is connected to the root of the last sub-cluster following the hierarchy of the clustering neglecting the fact that there are much cheaper opportunities. The negative effect on the solution quality is noticeable especially for leaf nodes but is not restricted to them.

A possibility to deal with this problem is to *free* the leaves from their strict membership to a specific cluster and to allow them to connect to the root of any cluster in the cheapest possible way. This usually leads to – also visually observable – better results.

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) [17] to further refine them, see Algorithm 5. 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 $N(\mu, \sigma^2)$.

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, respectively.

5. THE ODD-DIAMETER CASE

In case the diameter bound D is odd no single node forms the center of a BDMST but an edge, the *center edge*, connecting the two center nodes. This increases – when considering complete instance graphs – the number of tests to determine the best center from $\mathcal{O}(|V|^2)$ (when D is even) to $\mathcal{O}(|V|^3)$. Due to the large number of trees that have to be derived from a height-restricted clustering in the various steps of the clustering heuristic the complexity increase has a substantial impact on the runtime.

An opportunity to speed-up computations in the odd-diameter case is to reduce in a preprocessing step the number of potential center edges that are considered. Given a height-restricted clustering the center edge (building the backbone at level H) will connect to every root node of the clusters at level $H-1$. In case the Euclidean instance is sufficiently large the set of plausible center edges can be restricted to edges where both endpoints are within the rectangle spanned by these root nodes. Since the optimal roots for the clusters at level $H-1$ are not known they have to be heuristically determined. This can easily be done by computing diameter two stars for all these clusters and then choosing for each cluster the center of the cheapest star as representative for its root node. With increasing diameter bound the rectangle spanned by these nodes at level $H-1$ becomes smaller, thus the number of potential center edges is further reduced.

When searching for a value of parameter x in Equation (1) to determine a good set of initial cutting positions through the dendrogram using binary search the charts as shown in Fig. 3 (objective value of the BDMST over x) are almost identical for an odd diameter D and the even diameter $D-1$. That is, a good value of x for an even D usually is also an appropriate choice for $D+1$ making it unnecessary to derive odd BDMSTs with an center edge in this step.

Unfortunately, this strong connection between the odd-diameter D case and its even correspondent $D-1$ is no longer observable in the greedy randomized adaptive search procedure to refine the initially calculated cutting positions. Since the runtime increases dramatically especially when the diameter bound is tight we implemented the following additional variant: In general just BDMSTs with an even diameter $D-1$ are derived during the refinement process. Only in case a new best solution is found also a BDMST with the correct odd diameter D is computed based on the current HRC.

6. USAGE AS IMPROVEMENT METHOD

The heuristics presented so far to derive a good tree from a height-restricted clustering can also be used to locally improve solutions obtained from another (meta-)heuristic since from each valid solution a corresponding hierarchical clustering of the nodes can be determined easily.

Creating a simple height-restricted clustering as shown in Fig. 2(b) from a diameter constrained tree (c) is straightforward, only root nodes not already part of the leaves of the HRC (just node 9 in the figure) have to be treated separately. Every such root has to be assigned to one of its direct sub-clusters where it additionally has to be propagated to further sub-clusters until a leaf of the HRC is reached. The decision

Table 1: Averaged objective values over all 15 Euclidean Steiner tree instances of Beasley's OR-Library with 1000 nodes for various diameter bounds and (meta-)heuristics, the standard deviations are given parentheses. In addition, the averaged maximum running times of the clustering heuristics that were used as time limit for CBTC and RTC are listed.

	without VND					with VND			
D	CBTC	RTC	Cd^A	Cd^B	t$_{\max}$(C) [s]	RTC	Cd^B	ACO	t$_{\max}$(C) [s]
4	329.0261 (6.02)	146.4919 (3.88)	68.3241 (0.72)	**68.3226** (0.70)	2.54 (0.09)	65.2061 (0.55)	**65.1598** (0.56)	65.8010 (0.48)	5.56 (1.01)
6	306.2655 (9.02)	80.8636 (2.40)	47.4045 (4.85)	**47.1702** (4.61)	4.55 (0.49)	41.4577 (0.36)	**41.3127** (0.50)	42.1167 (0.26)	9.94 (1.52)
8	288.3842 (7.52)	53.2535 (1.33)	37.0706 (1.35)	**36.9408** (1.34)	5.92 (0.42)	35.0511 (0.35)	**34.2171** (0.29)	34.7489 (0.23)	11.61 (1.61)
10	266.3665 (9.01)	41.1201 (0.68)	33.5460 (0.67)	**33.3408** (0.66)	6.79 (0.42)	32.1181 (0.31)	**30.9704** (0.24)	31.0388 (0.20)	13.43 (2.16)
12	250.0016 (8.01)	35.7590 (0.47)	32.2571 (0.48)	**31.9561** (0.44)	7.11 (0.33)	30.2897 (0.29)	29.1796 (0.26)	**28.6356** (0.23)	14.68 (2.49)
14	237.1403 (6.28)	33.3644 (0.30)	31.3790 (0.37)	**31.0176** (0.33)	7.00 (0.64)	29.0940 (0.28)	28.0093 (0.23)	**26.6524** (0.32)	15.05 (3.00)
16	224.3123 (5.72)	32.1965 (0.24)	30.7937 (0.33)	**30.4287** (0.29)	7.20 (0.72)	28.2433 (0.28)	27.1363 (0.19)	**25.5760** (0.19)	15.63 (2.89)
18	210.9872 (7.63)	31.5826 (0.24)	30.5182 (0.29)	**30.1348** (0.27)	7.32 (0.81)	27.6008 (0.27)	26.5601 (0.20)	**24.8811** (0.16)	16.78 (3.61)
20	197.1772 (7.99)	31.2682 (0.22)	30.3116 (0.31)	**30.0384** (0.28)	7.57 (0.76)	27.1091 (0.26)	26.1079 (0.23)	**24.3698** (0.15)	18.54 (3.89)
22	183.0157 (8.03)	31.0864 (0.22)	30.2344 (0.30)	**30.0739** (0.28)	8.56 (0.98)	26.6984 (0.28)	25.8048 (0.21)	**24.0129** (0.17)	21.39 (5.19)
24	172.8251 (10.59)	30.9921 (0.23)	**30.0202** (0.23)	30.1603 (0.27)	8.28 (1.41)	26.3648 (0.27)	25.4523 (0.24)	**23.7723** (0.20)	21.36 (6.42)
5	241.3032 (5.09)	117.3238 (2.22)	62.2867 (0.76)	**62.0646** (0.67)	24.59 (2.02)	58.9883 (0.53)	**58.7930** (0.56)	59.5964 (0.49)	30.82 (3.28)
7	222.1441 (4.50)	67.7577 (1.31)	46.7291 (3.92)	**46.4112** (3.73)	27.94 (1.79)	39.4703 (0.34)	**39.3817** (0.46)	39.9948 (0.25)	38.79 (4.03)
9	204.6141 (6.00)	47.3168 (0.85)	37.0224 (1.25)	**36.8904** (1.27)	18.27 (1.68)	33.9677 (0.30)	**33.2142** (0.25)	33.5907 (0.23)	32.51 (4.88)
11	189.7513 (4.62)	38.4754 (0.50)	33.4140 (0.70)	**33.1749** (0.66)	13.97 (0.71)	31.3661 (0.29)	30.3683 (0.20)	**30.2701** (0.19)	29.47 (4.70)
13	175.7382 (4.23)	34.5154 (0.32)	32.1094 (0.43)	**31.8041** (0.41)	12.79 (1.17)	29.7644 (0.28)	28.7554 (0.21)	**28.1224** (0.20)	29.94 (6.28)
15	163.1926 (4.31)	32.7069 (0.25)	31.2654 (0.35)	**30.8941** (0.32)	11.03 (1.27)	28.6966 (0.26)	27.6899 (0.20)	**26.3893** (0.25)	28.54 (6.29)
17	149.9852 (5.14)	31.8467 (0.23)	30.7699 (0.33)	**30.3664** (0.30)	8.93 (0.94)	27.9309 (0.27)	26.9097 (0.19)	**25.3794** (0.23)	28.47 (6.19)
19	139.9730 (4.32)	31.4048 (0.21)	30.5350 (0.29)	**30.0837** (0.27)	7.91 (1.08)	27.3691 (0.26)	26.3784 (0.20)	**24.7705** (0.18)	29.67 (7.37)
21	128.1830 (4.90)	31.1697 (0.23)	30.3017 (0.30)	**30.0384** (0.27)	7.60 (0.71)	26.9015 (0.26)	25.9415 (0.20)	**24.3128** (0.18)	30.05 (6.74)
23	119.5551 (4.46)	31.0421 (0.22)	**30.0627** (0.24)	30.1166 (0.31)	6.96 (0.81)	26.5346 (0.27)	25.6021 (0.21)	**23.9719** (0.21)	28.55 (7.05)
25	110.6725 (4.39)	30.9772 (0.23)	**29.9450** (0.21)	30.1393 (0.24)	6.68 (0.89)	26.2126 (0.26)	25.2289 (0.21)	**23.7773** (0.25)	25.59 (6.02)

which sub-cluster to choose can be based on the same criterion as in the agglomeration process the choice which clusters to merge, i.e. a root node is assigned to the sub-cluster where the maximum distance to a node of it is minimal.

7. COMPUTATIONAL RESULTS

The experiments have been performed on an AMD Opteron 2214 dual-core machine (2.2GHz) utilizing benchmark instances already used in the corresponding literature (e.g. [15, 11]) from Beasley's OR-Library [3, 2] 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, where for each instance 30 independent runs have been performed, together with the standard deviations in parentheses. Considered are the two previous construction heuristics CBTC and RTC as well as the clustering heuristic C where each cluster a good root node is assigned using one of the two dynamic programming approaches d^A (restricted search space) and d^B (approximating optimal cluster roots using a correction value κ). Since d^A and d^B derive no optimal trees for a given clustering local improvement is used to further enhance their solutions.

Binary search to identify a good value for x was performed within $\frac{|V|}{2}$ and $|V|$, except when $D < 6$. In this latter case the interval bounds have been set to $\frac{|V|}{20}$ and $\frac{|V|}{8}$. 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 maximum running time of Cd^A and Cd^B, 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 suitable for this type of instances, its strength lies in problems with random edge costs. The clustering heuristic outperforms RTC for every diameter bound, where the gap in solution quality is huge when D is small and becomes less with increasing diameter bound. In general, Cd^B outperforms all other heuristics significantly with an error probability of less than $2.2 \cdot 10^{-16}$. Only when the diameter bound gets noticeably loose the first dynamic programming approach Cd^A dominates Cd^B (error probability always less than $2.13 \cdot 10^{-9}$). It can also be seen that in the even-diameter case the runtime of the clustering heuristic only increases moderately with the number of levels in the height-restricted clustering. When D is odd the search for a good center edge dominates the runtime. Thus, with increasing D the clustering heuristics even get faster since the number of potential center edges to be considered, determined in the presented preprocessing step, decreases (less direct successors of the root cluster in the HRC).

We also performed test where a strong variable neighborhood descend (VND) as proposed in [11] has been applied to the best solutions of the various construction heuristics. As expected, it flattens the differences but still the BDMSTs derived from clustering heuristic solutions are in general of higher quality. On instances with diameter bounds less than approximately 10, these trees – computed in a few seconds – can also compete with results from the leading metaheuristic, the ACO from [11], which requires computation times of one hour and more.

Experiments including the usage of the clustering heuristic as improvement method (see Section 6) in the VND revealed that it can significantly improve the solution quality, but only in case the diameter bound D is less than (roughly) 14 on instances with 1000 nodes. Otherwise, the VND with the four neighborhoods presented in [11] is usually able to reach the same solution quality, and this typically in less time.

8. CONCLUSIONS AND FUTURE WORK

On the more difficult to solve Euclidean BDMST instances fast construction heuristics proposed so far fail to compute a good backbone consisting of few but long edges to allow the majority of the nodes to connect to the tree via relatively short edges. In this work we presented a constructive heuristic that exploits a hierarchical clustering to guide the process of building a backbone.

The clustering heuristic constructs diameter constrained trees within three steps: determining a hierarchical clustering, reducing the height of this clustering according to the diameter bound, and finally deriving a BDMST from this height-restricted clustering. Various techniques are used within the individual phases, e.g. a GRASP-like strategy to refine cutting positions through the dendrogram, or dynamic programming to assign each cluster a good root node.

In particular on large Euclidean instances with more than 500 nodes 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. The computation of our heuristic followed by VND, however, requires only a few seconds in comparison to one hour and more per run of the ACO.

The negative effects when strictly following the clustering as discussed in Section 3.4 may be further addressed in two different ways. One simple approach would be to let the clustering-based construction heuristic build only the first part (near the center of the BDMST) of the backbone and to use a Prim based algorithm (CBTC or RTC) for the remaining nodes. A more sophisticated version would allow a root u of a sub-cluster not only to connect to the root of its direct parent cluster v but to any node of the already built backbone on the path from the center of the BDMST to u. In case a cheaper connection is possible than (u, v) some clusters merged in the subtree rooted at u for the height-restricted clustering can again be split since now u is connected at least one edge closer to the center of the BDMST, and so this subtree would otherwise not fully exploit the available height.

9. REFERENCES

[1] K. Bala, K. Petropoulos, and T. E. Stern. Multicasting in a linear lightwave network. In *Proc. of the 12th IEEE Conference on Computer Communications*, pages 1350–1358. IEEE Press, 1993.

[2] J. Beasley. OR-Library: Capacitated MST, 2005. `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/capmstinfo.html`.

[3] J. E. Beasley. A heuristic for Euclidean and rectilinear Steiner problems. *European Journal of Operational Research*, 58:284–292, 1992.

[4] R. E. Bellman. *Dynamic Programming*. Dover Publications Inc., 2003.

[5] A. Bookstein and S. T. Klein. Compression of correlated bit-vectors. *Information Systems*, 16(4):387–400, 1991.

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

[7] L. Gouveia and T. L. Magnanti. Network flow models for designing diameter-constrained minimum spanning and Steiner trees. *Networks*, 41(3):159–173, 2003.

[8] L. Gouveia, T. L. Magnanti, and C. Requejo. A 2-path approach for odd-diameter-constrained minimum spanning and Steiner trees. *Networks*, 44(4):254–265, 2004.

[9] M. Gruber and G. R. Raidl. (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.

[10] M. Gruber and G. R. Raidl. Cluster-based (meta-)heuristics for the Euclidean bounded diameter minimum spanning tree problem. In A. Quesada-Arencibia et al., editors, *Extended Abstracts of the Twelfth International Conference on Computer Aided Systems Theory (EUROCAST 2009)*, pages 228–231, Gran Canaria, Spain, 2009.

[11] M. Gruber, J. van Hemert, and G. R. Raidl. Neighborhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA, and ACO. In M. Keijzer et al., editors, *Proc. of the Genetic and Evolutionary Computation Conference 2006*, volume 2, pages 1187–1194, 2006.

[12] A. Jain, M. Murty, and P.J.Flynn. Data clustering: a review. *ACM Computing Surveys (CSUR)*, 31:264–323, 1999.

[13] B. A. Julstrom. Greedy heuristics for the bounded diameter minimum spanning tree problem. *Journal of Experimental Algorithmics (JEA)*, 14:1.1:1–1.1:14, February 2009.

[14] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[15] G. R. Raidl and B. A. Julstrom. Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In G. Lamont et al., editors, *Proc. of the ACM Symposium on Applied Computing*, pages 747–752. ACM Press, 2003.

[16] K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989.

[17] M. Resende and C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2003.

[18] A. Singh and A. K. Gupta. Improved heuristics for the bounded-diameter minimum spanning tree problem. *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, 11(10):911–921, 2007.