Fast Heuristics for Large Instances of the Euclidean Bounded Diameter Minimum Spanning Tree Problem

C. Patvardhan and V. Prem Prakash
Faculty of Engineering, Dayalbagh Educational Institute, Agra - 282005, India
E-mail: cpatvardhan@gmail.com, vpremprakash@acm.org

A. Srivastav
Institut für Informatik, Christian-Albrechts-Universität zu Kiel, Kiel, Germany
E-mail: asr@informatik.uni-kiel.de

Keywords: heuristic, euclidean, bounded diameter minimum spanning tree, constrained diameter, greedy

Received: January 29, 2015

Given a connected, undirected graph G = (V, E) on n = $|V|$ vertices, an integer bound D $\geq$ 2 and non-zero edge weights associated with each edge e $\in$ E, a bounded diameter minimum spanning tree (BDMST) on G is defined as a spanning tree T $\subseteq$ E on G of minimum edge cost w(T) = $\sum$ w(e), $\forall$ e $\in$ T and tree diameter no greater than D. The Euclidean BDMST Problem aims to find the minimum cost BDMST on graphs whose vertices are points in Euclidean space and whose edge weights are the Euclidean distances between the corresponding vertices. The problem of computing BDMSTs is known to be NP-Hard for 4 $\leq$ D $< n - 1$, where D the diameter bound. Furthermore, the problem is known to be hard to approximate. Heuristics are extant in the literature which build low cost, diameter-constrained spanning trees in O(n^3) time. This paper presents some fast and effective heuristic strategies for the Euclidean BDMST Problem and compares their performance with that of the best known existing heuristics. Two of the proposed heuristics run in O($n^2\sqrt{n}$) time and another faster heuristic runs in O(n^2), thereby allowing them to quickly build low cost BDSTs on larger sized problems than have been attempted hitherto. The proposed heuristics are shown to perform better over a wide range of benchmark instances used in the literature for the Euclidean BDMST Problem. Further, a new test suite of much larger problem sizes than attempted hitherto in the literature is designed and results presented.

Povzetek: Podan je hevristični postopek za hitro gradnjo minimalnega prekrivnega drevesa.

1 Introduction

Given a connected, weighted, undirected graph G and a positive integer D, the Bounded-Diameter Minimum Spanning Tree (BDMST) problem seeks a low cost spanning tree from amongst all spanning trees of G containing paths with no more than D edges. Formally, a bounded-diameter spanning tree (BDST) is a spanning tree T $\subseteq$ E on G = (V, E), whose diameter is no greater than D. The BDMST problem aims to find a bounded diameter spanning tree of minimum cost w(T) = $\sum$ w(e), $\forall\ e \in T$. Restricting the problem to Euclidean graphs (where vertices are points in Euclidean space and edge weights are the Euclidean distances between pairs of vertices) gives rise to the Euclidean BDMST Problem [1]. The problem finds application in several domains, ranging from distributed mutual exclusion [2] to wire-based communication network design [3] and data compression for information retrieval [4]. An important application also occurs in reducing the source-sink delays and total wire length in VLSI routing. Barring the special cases of D = 2, D = 3, D = $n - 1$, and all edge weights being the same, the BDMST Problem is known to be NP-Hard [5]. Furthermore, the problem is also known to be hard to approximate; it has been shown that no polynomial time approximation algorithm can be guaranteed to find a solution whose cost is within log (n) of the optimum [6].

An exact algorithm for the BDMST Problem is given by Achuthan and Caccetta in [7]. This is improved by Achutan et al. [8], wherein a branch and bound framework is given which utilizes different branching rules and simple heuristics. Gouveia and Magnanti [9], give several variants of multi-commodity flow (MCF) formulations for the BDMST problem which achieve extremely tight LP bounds (within 1% of the optimal solution for almost all benchmarks tested). However, this approach has been able to solve BDMST instances of up to only 60 node graphs to optimality. In general, the exponential time complexity of exact algorithms allows them to solve only very small problem instances; this motivates the search for fast heuristics and meta-heuristic techniques which can approximate low cost BDSTs on much larger problem sizes within reasonable time.

Several meta-heuristics are given in the literature that evolve BDMSTs on larger problem instances, including an ant colony algorithm [18], evolutionary algorithms

[20],[21],[22] and a recent learning automata-based algorithm [23].

Many of the best known existing heuristics for the BDMST problem are based on a greedy, Prim's [19] algorithm-like strategy. Each of these heuristics works well under certain conditions in the Euclidean BDMST case, for instance when the range of diameter bounds is restricted to a small range or when the diameter bound is very small. Further, none of the existing heuristics is suitable for working on very large sized problems, as they require too much computation time for building BDSTs on large problem instances. A key goal of the research presented in this paper has thus been to develop fast and robust heuristics that would build low cost BDSTs on very large problem sizes. The extant heuristics in the literature are briefly discussed as follows.

Ayman Abdalla and Narsing Deo describe in [10], several construction heuristics for the BDMST problem, including a Prim's [19] algorithm–based heuristic called the one-time tree construction (OTTC) heuristic that runs in O(n^4) time and produces low cost BDSTs when the diameter bound D is small. Abdalla and Deo also give two iterative refinement algorithms that start with an unconstrained MST and iteratively decrease the length of long paths until the diameter constraint is satisfied. The center-based tree construction (CBTC) heuristic given by Julstrom in [11] performs better than OTTC both in terms of solution quality and running time (it requires O(n) time less than OTTC) by constructing the BDST as a height-restricted tree rooted at a central node (or two nodes if the diameter is odd). A randomized tree construction heuristic (RTC) is also given in [11] wherein each next node to be appended to the BDST is chosen at random and appended greedily.

The RTC and CBTC heuristics are improved further by Singh and Gupta [12] and Singh and Saxena [13]. The improved heuristics given in [12] are called RGH-I and CBTC-I, which try to improve RTC and CBTC in terms of both speed and solution quality. In particular, for each vertex v (excluding the root and vertices adjacent to the root) the heuristics search for tree vertices of lower depth than v to which v can be appended at lesser cost. Singh and Saxena [13] improve these heuristics further and demonstrate their effectiveness on a standard set of problem instances used widely in the literature.

A hierarchical clustering-based heuristic for the Euclidean BDMST problem is given by Gruber and Raidl [14], which obtains low cost BDSTs when the diameter constraint is very small.

The Center-based Least Sum-of-Costs (CBLSoC) heuristic given by Patvardhan et al.[15] builds a low cost BDST by repeatedly appending the non-tree vertex with the lowest mean cost to all the remaining non-tree vertices in the graph. This heuristic is run starting from every graph vertex and returns the lowest cost BDST obtained. It has a running time of O(n^3) and performs competitively vis-a-vis the other heuristics. Parallel versions of the CBTC, RTC and CBLSoC heuristics are given in Patvardhan et al.[16] and their performance compared over a comprehensive set of Euclidean and random benchmark graphs.

This paper presents some fast heuristics for the Euclidean BDMST problem. The first of these is a variant of the CBLSoC heuristic [15]. This heuristic, called CBLSoC-lite, produces BDSTs with comparable/better (lower) costs as compared to existing heuristics on a wide range of benchmarks. Two other "Quadrant-Centers based" heuristics try to construct an effective backbone of a small number of low height nodes appended to the tree via relatively longer edges, and then build the rest of the BDST. The heuristics presented in this work typically take less time to build a low cost BDST vis-à-vis extant heuristics. This allows them to handle much larger problem sizes than attempted hitherto by any other heuristic. Their performance is demonstrated on a test suite of completely connected Euclidean graphs having up to 10,000 vertices.

Subsequent sections of the paper are organized as follows: section 2 discusses three well known heuristics for the problem (OTTC, CBTC and RTC), section 3 describes CBLSoC and the proposed CBLSoC-lite heuristic, and Section 4 presents the proposed "Quadrant centers-based" heuristic strategy. Computational results obtained on benchmark problem instances and other larger randomly generated graphs are presented and summarized in section 5, and concluding remarks are made in section 6.

2 Some well known heuristics

This section presents several well known heuristics for the BDMST Problem and summarizes their key characteristics.

2.1 One-time Tree Construction (OTTC) Heuristic

One-Time Tree Construction (OTTC) given by Abdalla et al. [10] is a greedy heuristic that computes the diameter of the spanning tree at each step and ensures that the incoming vertex does not violate the diameter constraint. In order to obtain a low cost BDST, the OTTC algorithm is run starting from every vertex of the graph. For each vertex that it starts from, OTTC repeatedly appends to the growing MST the lowest-cost edge that appends a new vertex to the tree without violating the diameter bound. Adding each new vertex also involves updating the path lengths and eccentricities of the tree vertices, requiring at most O(n^2) time. This is done n-1 times in each run, so the algorithm has a total running time of O(n^3). As the algorithm is run starting once from each graph vertex (i.e., totally n times), the total time complexity is O(n^4). Pseudo-code for this heuristic is given in listing 1.

Listing 1: OTTC heuristic

```
V[.] ← set of graph vertices
S[.] ← set of tree vertices, initially empty
for each $v \in V$ do
    T ← { }
    S ← { }
    S ← S ∪ {v}
    while $|T| \neq (n-1)$ do
        Choose x ∈ V \ S and u ∈ S such that $cost(u,x)$ is minimal ∀ u ∈ S and diameter constraint D is not violated
        $T \leftarrow T \cup (u,x)$
        S ← S ∪ {x}
    if cost(T) < cost(bestTree) then
        bestTree ← T
return bestTree
```

Listing 2: CBTC heuristic

```
U[.] ← set of unconnected graph vertices
C[.] ← set of tree nodes with depth < ⌊D/2⌋ for each vertex $v_0 \in V$ do
    Set $v_0$ as root
    U ← U \ {$v_0$}
    C ← {$v_0$}
    depth[$v_0$] = 0
    if D is odd then
        Choose vertex $v_1$ ∈ U such that cost($v_0$, $v_1$) is minimal
        U ← U \ {$v_1$}
        C ← {$v_1$}
        depth[$v_1$] = 0
        T ← T ∪ ($v_0$, $v_1$)
    while U ≠ {} do
        Find u ∈ C and v ∈ U such that cost(u,v) is minimal
        T ← T ∪ (u, v)
        U ← U \ {v}
        depth[v] ← depth[u] + 1
        if (depth[v]<⌊D/2⌋) then
            C ← C ∪ {v}
return the tree with lowest cost out of all trees generated above
```

2.2 Center-based tree construction (CBTC) heuristic

In a tree with diameter D, no vertex is more than D/2 hops or edges from the root vertex of the tree [17]. This motivates a faster Prim's algorithm-based heuristic called Center-Based Tree Construction (CBTC) heuristic [11], which improves OTTC by building the BDST from the tree's center, keeping track of the depth of each tree node and ensuring that no node depth exceeds $\lfloor D/2 \rfloor$. This heuristic avoids the task of repeatedly computing the tree diameter before appending each node, and returns the lowest cost BDST obtained over n runs, each starting from a different graph vertex, thereby bringing down the total running time to O(n^3). Pseudo-code for the CBTC heuristic is given in listing 2.

2.3 Randomized tree construction (RTC) heuristic

In the Randomized Tree Construction Heuristic (RTC), the center of the spanning tree is chosen at the outset as one vertex (if D is even) or two connected vertices (if D is odd) randomly selected from the set of graph nodes. Each next vertex is then chosen at random and connected to the tree greedily such that the inclusion does not yield a tree of diameter greater than the diameter bound D. Building the BDST requires repeating this process through $n-1$ iterations. As before, this process is repeated n times, and the lowest cost BDST is returned. Hence the total running time of this heuristic is O(n^3). Pseudo-code for this heuristic is given in listing 3.

2.4 Some Other Heuristics

Singh and Gupta [12] improve the CBTC and RTC heuristics in two ways. Firstly, after building a BDST using either construction heuristic, it is checked for each vertex v∈V in the BDST whose depth is greater than 1 (essentially covering all vertices that are not either the root(s) or the vertices immediately connected to the root(s)) whether it can be reattached to a BDST vertex of depth less than depth[v] via a lower cost edge. Secondly, the heuristics maintain a sorted cost matrix and searching for a low cost edge to append a vertex v to the BDST in the $n/10$ elements of the cost matrix row entry for the vertex v. These two heuristics are further improved in Singh and Saxena [13], where they are called RGH+HT and CBTC+HT respectively, by allowing a sub-tree rooted at v to be connected to any vertex of the tree irrespective of its depth, provided the cost is reduced and the feasibility of the resulting BDST is retained. Gruber and Raidl [14] use agglomerative hierarchical clustering to guide the creation of an effective BDST backbone and transform the resulting dendogram structure into a height-restricted clustering that satisfies the diameter constraint. The heuristic then uses either a greedy heuristic or one of two dynamic programming (DP) approaches to identify a good root node within each cluster. The first DP approach restricts the search space of root nodes of a cluster to the root nodes of sub-clusters, while the second approximates optimal cluster roots using a correction value for estimating the cost of connecting each graph vertex as a leaf node of the BDST. The dynamic programming approaches take $O(H.|V|^2)$ and $O(|V|.|E| + H.|V|^2)$ time, when D is even and odd respectively, for computing the roots of clusters, where $H-1$ is the tree height. The re-

Listing 3: RTC heuristic

```
U[.] ← set of unconnected graph vertices
C[.] ← set of tree nodes with depth < ⌊D/2⌋ for n times do
    Set as root, a random vertex $v_0 \in$ U
    U ← U \ {$v_0$}
    C ← {$v_0$}
    depth[$v_0$] = 0
    if D is odd then
        Choose another random vertex $v_1 \in$ U
        U ← U \ {$v_1$}
        C ← C ∪ {$v_1$}
        depth[$v_1$] = 0
        T ← T ∪ ($v_0$, $v_1$)
    while U ≠ {} do
        Choose a random vertex v ∈ U
        Find u ∈ C such that cost(u,v) is minimal
        T ← T ∪ (u, v)
        U ← U \ {v}
        depth[v] ← depth[u] + 1
        if (depth[v]<⌊D/2⌋) then
            C ← C ∪ {v}
return the tree with lowest cost out of all trees generated above
```

sults of the heuristic are further improved using a variable neighborhood descent (VND) given in [18].

2.5 Discussion

A major drawback of the OTTC and CBTC heuristics is that they always use low cost edges to build the tree, thus necessitating the later vertices to be appended to the tree through higher cost edges. This often results in BDSTs with larger costs, especially when the diameter bound is small. One way to overcome this is to select each node randomly and then append it to the tree greedily, as is done by the RTC heuristic. Possibly due to the random nature of node selection, the initial "backbone" of the BDST often turns out better in RTC than in the OTTC and CBTC heuristics, leading to lower cost BDSTs when the diameter bounds are small. However, the performance of the RTC heuristic rapidly deteriorates as the diameter bound is increased, as discussed in section 5. The heuristics of Singh and Saxena [13] produce lower cost BDSTs than CBTC and RTC. The Hierarchical Clustering-based heuristic produces very low cost BDSTs, but is seen to be effective only when diameter bounds are small.

3 The CBLSoC-lite and CBLSoC Heuristics

The Center-based Least Sum-of-Costs *Lite* (CBLSoC-lite) heuristic starts by selecting as root the vertex (or two vertices, in case of odd diameter) with lowest mean cost to all other graph vertices. Thereafter, each new graph vertex selected has the lowest sum of costs to all the remaining graph vertices. This vertex is then appended to the tree greedily via the lowest cost edge that does not violate the diameter bound. The heuristic uses a center-based approach, building the BDST from the tree's center, keeping track of the depth of each tree node and ensuring that no node depth exceeds ⌊D/2⌋. This preempts the need for dynamically computing the spanning tree's diameter at each step and results in a total computational time of $O(n^2)$ for the heuristic. Pseudo-code for the heuristic is given in listing 4. The CBLSoC heuristic iteratively builds BDSTs starting once from each graph vertex and returns the lowest cost BDST thus obtained. This results in further improvements in BDST costs, but incurs an additional overhead of O(n), and hence a total running time of O(n^3) for the heuristic.

Listing 4: CBLSoC-LITE heuristic

```
U[.] ← Adjacency matrix containing edge weights of the graph G
C[.] ← Set of unconnected graph vertices
Choose u∈V such that sum of entries in row u of adj. matrix A is minimal
U ← U \ {u}
if D is odd then
    choose v ∈ V \ {u} such that sum of entries in row v of A is minimal
T ← T ∪ (u,v) U ← U\{v}
while U ≠ {} do
    Choose x∈V \ T such that $\sum$ cost(x,y) is minimal ∀ y∈V\T,y≠x and diameter constraint is not violated
    Choose u such that cost (u, x) is minimal, ∀ u ∈ T
    T ← T ∪ (u, x)
    U ← U \ {x}
return T
```

The CBLoC-lite and CBLSoC heuristics produce better (lower) cost BDSTs in comparison to the OTTC, CBTC and RTC heuristics, as shown in section 5 on a large number of benchmark problems.

4 *Quadrant Centers*-based Heuristics

As discussed in section 2, the greediness inherent in the OTTC and CBTC heuristics causes the backbone of the growing BDST to be typically constituted of short edges, thus forcing several nodes to be appended using long edges and thereby increasing the total cost. The relatively less greedy, center-based LSoC heuristics discussed in section 3 mitigate this problem to some extent, as shown in the experimental results presented in section 5.

Another group of heuristics is proposed which start by empirically fixing the tree center and adding a few graph vertices to the tree, thereby building a backbone comprising of a small number of nodes connected to the tree via relatively longer edges. The remaining nodes are then appended to the BDST either greedily or by using the CBLSoC heuristic.

Listing 5: QCH-GREEDY heuristic

```
Choose $v_0 \in$ V | { $\sum$ cost($v_0$, x) is minimal $\forall$ x $\in$ V, x $\neq v_0$ }
U $\leftarrow \{v_0\}$
if D is odd then
    Choose $v_1 \in$ U | {cost($v_0$, $v_1$) is minimal, $\forall$ $v_1 \in$ U }
    T $\leftarrow$ T $\cup$ ($v_0$, $v_1$)
    U $\leftarrow$ U $\cup$ $\{v_1\}$
for qsize = 2 to $\sqrt{n}$ do
    for i = 1 to qsize do
        Choose p $\in$ quadrant i|{ $\sum$ cost(p,j) is minimal, where p, j $\in$ U, j $\in$ quadrant i and j $\neq$ p }
        T $\leftarrow$ T $\cup$ (k, p), where cost(k, p) is minimal of all tree vertices k
        U $\leftarrow$ U $\setminus$ {p}
    Append each remaining node x$\in$U to T greedily via the lowest cost edge such that depth[x] $\leq \lfloor D/2 \rfloor$
    for each vertex i $\in$ T do
        $\forall$ j $\in$ T such that depth[j] $< \lfloor D/2 \rfloor$, if cost(i,j) < cost (i, parent[i]), replace (i,parent[i]) with (i,j) in the BDST
    bestT $\leftarrow$ lowest cost BDST found so far
return bestT
```

The Euclidean problem domain is widely modeled in the literature as a set of real points normally distributed in the unit square. Using this model, the proposed heuristics build a tree "backbone" by first choosing the root node of the BDST using the CBLSoC heuristic. Specifically, the node with the lowest mean cost to all other graph nodes is added to the BDST and chosen as the central, or root vertex. If (the diameter) D is even, then this single node serves as the root of the BDST. If D is odd, then the non-tree node with the lowest mean cost to the remaining graph vertices is also selected and added to the tree via the lowest cost edge; the sub-graph comprising these two nodes and the edge joining them is now considered as the center of the BDST. The remaining graph vertices are segregated into the "quadrants" of a uniform MxM matrix in the unit square, $2 \leq M \leq \sqrt{N}$ (each element of this matrix is termed a quadrant, for want of a more suitable expression). Within each quadrant, the node with lowest mean cost to all other nodes within the same quadrant is set as a tree backbone node of depth 1. The remaining vertices are then appended to the tree either greedily (this is called the QCH-Greedy heuristic) or using the CBLSoC heuristic in selecting each next vertex to append to the tree (this is called QCH-LSoC heuristic), while ensuring that the diameter constraint is not violated.

Listing 6: QCH-LSOC heuristic

```
Choose $v_0 \in$ V | { $\sum$ cost($v_0$, x) is minimal $\forall$ x $\in$ V, x $\neq v_0$ }
U $\leftarrow \{v_0\}$
if D is odd then
    Choose $v_1 \in$ U | {cost($v_0$, $v_1$) is minimal, $\forall$ $v_1 \in$ U}
    T $\leftarrow$ T $\cup$ ($v_0$, $v_1$)
    U $\leftarrow$ U $\cup$ $\{v_1\}$
for qsize = 2 to $\sqrt{n}$ do
    for i = 1 to qsize do
        Choose p $\in$ quadrant i|{ $\sum$ cost(p,j) is minimal, where p, j $\in$ U, j $\in$ quadrant i and j $\neq$ p }
        T $\leftarrow$ T $\cup$ (k, p), where cost(k, p) is minimal of all tree vertices k
        U $\leftarrow$ U $\setminus$ {p}
    for remaining vertices in U do
        Choose k $\in$ U | { $\sum$ cost(k,j) $\forall$ j$\in$U, j$\neq$k} is minimal
        T $\leftarrow$ T $\cup$ (k,x), where (k,x) is the lowest cost edge s.t. depth[k] $\leq \lfloor D/2 \rfloor$
        U $\leftarrow$ U $\setminus$ {p}
    for each vertex i $\in$ T do
        $\forall$ j $\in$ T such that depth[j] $< \lfloor D/2 \rfloor$, if cost(i,j) < cost (i, parent[i]), replace (i,parent[i]) with (i,j) in the BDST
    bestT $\leftarrow$ lowest cost BDST found so far
return bestT
```

Pseudo-code for these two heuristics is given in listings 5 and 6 respectively.

Both the heuristics attempt to find an effective backbone by varying the number of quadrants up to n (where n is the input size) and building the backbone accordingly. The heuristics return the lowest cost BDST obtained by this procedure.

Setting up the backbone of the BDST requires O(n) time in the greedy QCH heuristic (QCH-Greedy) and $O(n^2)$ time in the CBLSoC-based variant (QCH-LSoC). In the greedy variant, the process of greedily appending each node to the BDST requires O(n) time, resulting in a total running time of $O(n^2)$. Running the heuristic up to $\sqrt{n}$ times in the worst case gives a total running time of $O(n^2\sqrt{n})$.

In the LSoC-based variant, identifying the non-tree node with the lowest mean cost to all other non-tree nodes can be achieved in O(n) time when we keep track of the mean costs from each such node to all other such nodes and update in linear time, and appending it greedily to the BDST would also require linear time. Thus the total running time for the LSoC-based QCH heuristic is also O($n^2\sqrt{n}$). In practice, the heuristics terminate when there is no further improvement in cost as compared to the previous iteration. Both heuristics attempt to reattach each node of the BDST (excepting the root nodes) at a lower cost, if possible, in a simple post-processing step that requires O(n^2) time, which does not result in any change in the overall time complexity, and slightly improves the tree cost in several cases.

5 Comparison of performance on benchmarks

The Euclidean Steiner Problem data sets given in Beasley's OR-Library[1] have been used extensively in the literature for benchmarking heuristics and algorithms for the BDMST Problem. These instances comprise of vertices placed at random in the unit square, fifteen instances of each size for graphs of up to 1000 vertices. Julstrom [9] uses an enhanced test suite of Euclidean problem instances that augments the OR-Library instances with randomly generated Euclidean graphs, fifteen each of 100, 250, 500 and 1000 vertices, whose edge weights are, as before, the Euclidean distances between (randomly generated) points in the unit square. Another test suite of larger Euclidean problem instances comprising of thirty randomly generated Euclidean graphs of 1500, 2500, 5000 and 10,000 vertices was developed by the authors for comparing the performance of the heuristics presented in this work on larger problem instances. These problems are referred to in this paper as large problem instances, to differentiate from the enhanced test suite of standard sized problems given by Julstrom [9].

The heuristics presented in this paper were tested on the thirty "standard" problem instances of 100, 250, 500 and 1000 vertex graphs provided in the enhanced benchmark suite of [9], totaling 120 completely connected Euclidean graphs, and the mean (X) and standard deviation (SD) of tree costs, and mean CPU times (t_{avg}) were obtained for each node size. The results obtained on the enhanced benchmark suite of standard problem instances for the OTTC, CBTC, RTC, CBLSoC-lite, CBLSoC and proposed QCH heuristics are given in table 1.

The heuristics were also tested on the thirty "large" problem instances of 1500, 2500, 5000 and 10000 vertex graphs; the mean and standard deviation of tree costs and mean CPU times obtained for all the heuristics are given in table 2. Results for the OTTC heuristic were not computed for larger sized problems because it takes too much computational time, as is obvious from the times given in table 1 for 1000 vertex graphs. In any case OTTC always performs worse than the CBTC heuristic.

The values used for D (the diameter bound) in all the tests were always less than the smallest diameter of an unconstrained MST on each set of graphs. The mean CPU times quoted in table 1 for the OTTC heuristic were obtained in [9] on a Pentium IV, 2.53 GHz processor with 1 GB memory. All the other heuristics were implemented in C on a Dell Precision T-5500 Workstation with 12 Intel Xeon 2.4-Gigahertz processor cores and 11 GB RAM running Red Hat Enterprise Linux 6.

The proposed heuristics were compared in terms of lowest and mean BDST costs obtained and computation time vis-a-vis the improved heuristics of Singh et al. [13] on Euclidean problem instances in table 3 and the hierarchical clustering-based heuristics of Gruber and Raidl [1] in table 4.

The mean BDST costs for the CBTC, RTC, CBLSOC-lite and CBLSoC heuristics given in table 1 show that the CBLSoC-lite and CBLSoC heuristics outperform OTTC on all problem instances and produce lower mean costs vis-à-vis the CBTC heuristic on most instances.

The RTC heuristic produces relatively lower cost trees, but this is only when the diameter bound is very small; as the diameter bound is increased the lowest cost BDSTs are the ones produced by CBLSoC-lite and CBLSoC. In order to understand this behavior, we observe that the OTTC and CBTC heuristics always greedily append to the tree, the node with the lowest cost to the tree. As a result the tree backbone ends up comprising of a small number of relatively short edges, forcing many of the remaining graph vertices to be appended via longer edges in order to maintain the diameter bound, resulting in higher tree costs. In a sense, the inherent greediness of the heuristic adversely affects its performance.

The RTC heuristic, possibly due to its randomized node selection approach, has a much better chance of building a tree backbone close to clusters of nodes, several of which might then be appended to the backbone using short edges. When D is small, it usually returns trees with lower costs than any of the other heuristics (OTTC, CBTC or CBLSoC-lite). However, as the diameter bound is increased, the RTC policy of always choosing the next node to append in a random manner leads to several poor choices, thereby contributing adversely to the overall BDST cost. The heuristic fails to produce any improvements in BDST costs with as the diameter bound is relaxed and is eventually surpassed in performance by the other heuristics.

The CBLSoC-lite heuristic is relatively less greedy in the sense that the next node chosen to be appended to the tree is always the one with the lowest mean cost to all remaining nodes in the tree; this node need not necessarily be the node with the lowest cost to the tree. The performance of this heuristic, especially in terms of speedup, is significant. For instance, table 1 shows that while the OTTC and RTC average about 173 seconds and 15 seconds respectively for computing BDMSTs on the 1000 node instances,

[1]Maintained by J.E. Beasley, Department of Mathematical Sciences, Brunel University, UK. (http://people.brunel.ac.uk/ mastjjb/orlib/files)

the CBLSoC-lite heuristic takes about 0.03 seconds on average for problems of the same size, and computes BDMSTs with mean costs that are usually better. The CBLSoC heuristic takes O(n^3) time but produces lower cost BDSTs than CBLSoC-lite.

The QCH heuristics start by trying to fix up a "good" backbone for the BDST. The greedy variant, QCH-greedy incorporates the greedy selection strategy used by OTTC, CBTC and RTC; the other proposed variant use the node selection strategy followed by the CBLSoC heuristic. The QCH heuristics produce low cost BDSTs in general, as can be seen from the mean tree costs given in table 1.

The two QCH variants obtain competitive mean tree costs, with the QCH-greedy heuristic producing slightly better BDSTs on larger diameter bounds. The BDST costs obtained by the QCH heuristics are lower than that obtained by the RTC heuristic on all problem instances, more so when the diameter bounds are small. Both heuristics produce significantly lower cost BDSTs vis-à-vis CBLSoC and CBLSoC-lite when the diameter is small, and give competitive results on larger diameter bounds; the QCH variants perform better than OTTC and CBTC on all problem instances and for almost all diameter constraints.

On the large Euclidean problem instances, the computational time requirements of OTTC, CBTC, RTC and CBLSoC heuristics rapidly become prohibitively high (table 2), whereas the CBLSoC-lite and QCH heuristics are still able to quickly compute low cost BDSTs.

On 2500 vertex graphs for example, CBLSoC-lite computes lower cost BDSTs than CBTC on all except the smallest diameter bound, in less than 1/1000th of the time taken by CBTC (0.22 seconds for CBLSoC-lite versus 257.29 seconds for CBTC on the third 2500 vertex instance in table 2). On the same instance, the QCH-greedy heuristic computes the lowest cost BDST of all the heuristics in about 0.42 seconds.

RTC produces the lowest cost BDST of all the heuristics in instance each of 1500 and 2500 vertex graphs, and that too only on the smallest diameter bound. Furthermore it fails to improve tree costs as the diameter constraint is progressively increased. By contrast, CBLSoC-lite takes less than one second to build low cost BDSTs on benchmark graphs of the same size and outperforms CBTC and RTC as D is increased. Even on completely connected graphs of 10,000 vertices, the heuristic computes BDSTs in less than 5 seconds on average (the t_{avg} column in table 2).

As already observed with the standard sized Euclidean problems, the QCH heuristics obtain the lowest cost among the heuristics being compared. In particular, the QCH-LSoC heuristics almost always produce the lowest cost trees on smaller diameter bounds, with the QCH-greedy heuristic obtaining the lowest costs BDSTs on larger diameter constraints for most of the large problem instances. It is worth noting that the CBLSoC-lite and the two QCH heuristics compute low cost BDSTs in a fraction of the computation time taken by the other heuristics. This is illustrated in figure 1.

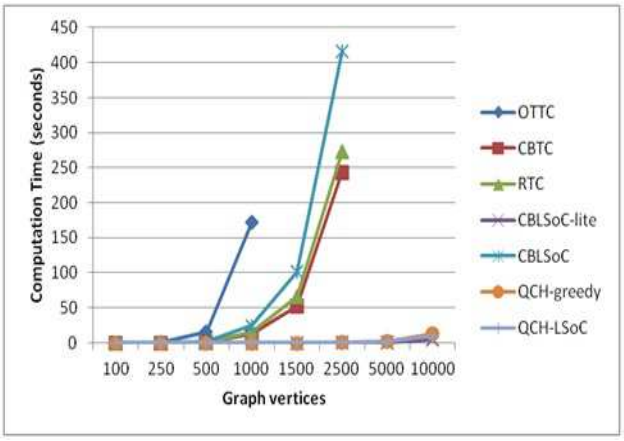

Figure 1: *Comparison of computational time taken by the heuristics*

The proposed heuristics are also compared with the improved heuristics of Singh and Saxena [13] in table 3 and with Gruber and Raidl's hierarchical clustering-based heuristic strategy in table 4. Both works provide computational results for small diameter bounds only on problem instances of up to 1000 vertex graphs.

Singh and Saxena [13] give the results obtained by their improved heuristics on the first five instances of the Beasley Euclidean Steiner Problem data sets for 50, 100, 250, 500 and 1000 vertex graphs, with diameter bounds of 5, 10, 15, 20 and 25 respectively.

Table 3 gives the lowest, mean and standard deviation (as applicable) of BDST costs obtained by these two heuristics in [13] vis-à-vis the proposed heuristics on Beasley's Euclidean problems. As the table shows, the faster CBLSoC-lite heuristic outperforms the CBTC+HT heuristic on smaller sized problems (50 and 100 node instances), and running this heuristic starting from each graph vertex (the CBLSoC heuristic) produces much better BDSTs than the CBTC+HT heuristic on all problem instances. When the diameter bound is very small, the lowest cost BDSTs are returned by the RGH+HT heuristic, which improves the results obtained by the RTC heuristic on these problem instances by about 8.26% on average. However, no further results on larger diameter bounds or on larger problem sizes are given in the literature for these heuristics. Further, the RGH+HT builds on the RTC heuristic, which has already been shown to perform poorly upon increasing the diameter bound on a wide range of benchmarks (tables 1 and 2). On the other hand, the CBLSoC and QCH heuristics are quite effective on larger diameter bounds and problem instances, as already demonstrated on Julstrom's enhanced test suite and the large problem instances.

Instances		**OTTC**			**CBTC**			**RTC**			**CBLSoC-lite**			**CBLSoC**			**QCH-Greedy**			**QCH-LSoC**		
n	**D**	**X**	**SD**	**t**	**X**	**SD**	**t**	**X**	**SD**	**t**	**X**	**SD**	**t**	**X**	**SD**	**t**	**X**	**SD**	**t**	**X**	**SD**	**t**
100	5	29.38	1.71	0.07	26.48	1.51	0.01	15.39	0.66	0.01	27.64	1.97	**<0.0001**	22.33	1.43	0.01	**13.76**	0.49	**0.0007**	**13.76**	0.49	**0.0007**
	10	18.43	1.86	0.07	15.59	1.28	0.01	9.76	0.29	0.01	16.61	1.84	**<0.0001**	12.54	0.96	0.01	9.53	0.35	**0.0003**	**9.43**	0.37	**0.0003**
	15	12.84	1.47	0.07	10.95	1	0.01	9.23	0.27	0.01	10.45	0.99	**<0.0001**	9.15	0.57	0.01	**8.47**	0.28	**0.0003**	8.63	0.28	**0.0003**
	25	8.06	0.59	0.07	7.69	0.34	0.02	9.16	0.25	0.01	7.42	0.3	**<0.0001**	**7.32**	0.24	0.01	7.89	0.25	**0.0003**	8.21	0.25	**0.0003**
250	10	58.2	5.38	1.18	49.07	2.99	0.1	16.84	0.31	0.08	53.82	5.3	**0.0013**	30.91	1.8	0.21	16.75	0.35	**0.0037**	**16.68**	0.48	**0.004**
	15	41.59	4.03	1.22	36.44	3.15	0.1	15.32	0.22	0.11	38.22	6.46	**0.0013**	21.79	1.79	0.23	**14.58**	0.28	**0.0037**	14.92	0.27	**0.0037**
	20	32.55	3.86	1.26	26.17	2.36	0.11	15.02	0.22	0.12	24.84	3.53	**0.0013**	17.84	1.13	0.21	**13.5**	0.31	**0.003**	13.9	0.42	**0.003**
	40	14.43	2.11	1.36	12.51	0.83	0.13	14.98	0.23	0.12	11.56	0.23	**0.0013**	**11.47**	0.2	0.2	12.17	0.38	**0.0027**	12.57	0.2	**0.0027**
500	15	106.87	5.03	12.69	94.38	5.56	0.82	22.21	0.33	0.85	100.67	6.51	**0.0053**	44.39	3.09	2.03	**22.64**	0.48	**0.0173**	22.7	0.51	**0.018**
	30	58.52	7.1	14.65	46.51	4.13	0.94	21.42	0.34	1.21	40.31	6.13	**0.006**	25.33	2.01	2.02	**18.23**	0.43	**0.013**	18.95	0.61	**0.0123**
	45	32.23	5.66	16.66	25.63	2.18	1.04	21.45	0.36	1.22	19.2	1.45	**0.0057**	17.8	0.78	2.19	**16.7**	0.48	**0.011**	18.02	0.69	**0.0117**
	60	20.33	3.46	17.64	17.72	0.92	1.16	21.42	0.34	1.03	16.13	0.32	**0.0053**	**16.03**	0.29	1.97	16.31	0.45	**0.011**	17.21	0.3	**0.0113**
1000	20	217.71	9.48	150.06	195.96	7.97	10.71	31.2	0.24	13.55	206.63	13.02	**0.0283**	51.65	1.7	23.66	**30.45**	0.52	**0.0797**	31.32	0.43	**0.0837**
	40	124.21	17.33	167.91	99.57	7.86	11.56	30.81	0.26	14.36	84.21	8.48	**0.031**	33.22	2.93	24.49	**25.23**	0.48	**0.0657**	26.51	0.64	**0.0503**
	60	69.83	12.2	183.21	50.97	5.24	12.94	30.81	0.26	16.06	31.72	3	**0.0323**	27.71	1.68	25.3	**23.49**	0.62	**0.053**	25.78	0.86	**0.0577**
	100	28.95	4.14	189.33	23.41	0.78	13.7	30.81	0.26	16.68	22.59	0.19	**0.0293**	**22.57**	0.19	24.61	22.59	0.5	**0.0527**	23.85	0.2	**0.0463**

Table 1: Results obtained on standard benchmark instances, thirty each of 100, 250, 500 and 1000 node graphs for the OTTC, CBTC, RTC, CBLSoC-lite, CBLSoC, QCH-Greedy and QCH-LSoC heuristics

Instances		**CBTC**			**RTC**			**CBLSoC-lite**			**CBLSoC**			**QCH-Greedy**			**QCH-LSoC**		
n	**D**	**x**	**SD**	**t**	**x**	**SD**	**t**	**x**	**SD**	**t**	**x**	**SD**	**t**	**x**	**SD**	**t**	**x**	**SD**	**t**
1500	30	119.73	14.19	45.03	**27.41**	0.4	58.44	235.92	50.2	**0.06**	41.6	4.53	97.71	35.19	1.3	**0.21**	33.03	1.33	**0.19**
	65	48.58	5.88	50.03	27.42	0.4	73.17	50.51	20.06	**0.06**	25.7	2.22	102.03	25.06	0.97	**0.17**	**24.64**	0.78	**0.12**
	100	30.09	2.86	56.02	27.39	0.4	60.76	23.81	2.2	**0.06**	21.73	0.69	104.28	**22.04**	0.97	**0.13**	22.71	0.73	**0.12**
	135	24.26	1.19	58.72	27.42	0.4	72.32	20.99	0.76	**0.06**	20.45	0.42	103.89	**20.84**	0.56	**0.12**	22.36	0.6	**0.11**
2500	40	182.05	18.47	208.39	**35.67**	0.38	273.07	345.22	60.71	**0.21**	50.42	5.61	395.42	43.78	2.26	**0.68**	40.71	1.85	**0.62**
	80	70.96	8.9	234.53	35.67	0.38	272.33	63.88	19.39	**0.22**	34.15	3.25	413.27	32.86	1.68	**0.52**	**31.69**	0.97	**0.47**
	120	43.31	4.63	257.29	35.67	0.38	272.53	31.98	3.68	**0.22**	29	2.05	416.9	**28.54**	1.02	**0.42**	29.65	1.42	**0.43**
	160	33.93	2.62	275.09	35.67	0.38	273.98	27.62	0.77	**0.22**	26.63	0.42	439.48	**26.69**	0.85	**0.38**	28.86	0.8	**0.42**
5000	50	-	-	-	-	-	-	800.67	138.84	**1.01**	-	-	-	63.72	1.85	**3.44**	**57.73**	2.72	**3.02**
	100	-	-	-	-	-	-	175.78	50.3	**1.07**	-	-	-	48.06	1.6	**2.72**	**45.23**	1.14	**2.47**
	150	-	-	-	-	-	-	47.35	6.22	**1.05**	-	-	-	**41.55**	1.58	**2.22**	41.76	1.39	**1.92**
	200	-	-	-	-	-	-	40.34	3.49	**1.06**	-	-	-	**38.49**	1.12	**1.9**	40.43	1.12	**1.96**
10000	60	-	-	-	-	-	-	1690.2	226.63	**4.76**	-	-	-	99.62	6.85	**15.61**	**88.74**	5.89	**12.46**
	120	-	-	-	-	-	-	404.44	139.71	**4.96**	-	-	-	71.45	5.47	**13.15**	**67.98**	2.45	**12.07**
	180	-	-	-	-	-	-	89.15	23.88	**5.01**	-	-	-	62.88	6.39	**11.31**	**60.63**	2.44	**9.76**
	240	-	-	-	-	-	-	60.88	4.44	**5.01**	-	-	-	**57.34**	2.56	**10.27**	58.17	2.93	**8.51**

Table 2: Results obtained on large Euclidean benchmark instances, thirty each of 1500, 2500, 5000 and 10000 node graphs for the CBTC, RTC, CBLSoC-lite, CBLSoC, QCH-Greedy and QCH-LSoC heuristics

Instances		CBTC+HT			CBLSoC-lite	CBLSoC			RTC			RGH+HT			QCH-Greedy	QCH-LSoC
N (D)	**No.**	**Best**	**Mean**	**SD**	**Best**	**Best**	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Best**	**Best**
50 (5)	1	13.28	21.8	5.33	11.83	**10.94**	**13.34**	**1.42**	9.34	12.82	2.48	**8.53**	12.56	2.14	8.63	8.63
	2	13.19	19.23	3.73	13.79	**11.75**	**12.86**	**0.61**	8.98	11.56	1.56	8.74	11.39	1.48	**8.26**	**8.26**
	3	11.59	19.06	3.82	11.59	**10.3**	**12.01**	**0.64**	8.76	11.54	1.9	**8.28**	10.66	1.21	8.33	8.33
	4	10.78	16.79	3.65	11.29	**9.83**	**10.96**	**0.52**	7.47	10.57	1.66	**7.54**	9.8	1.52	7.93	7.93
	5	12.31	18.3	3.28	12.28	**10.55**	**12.4**	**1.13**	8.79	10.91	1.61	**8.59**	10.49	1.48	8.84	8.84
100 (10)	1	17.34	28.6	7.09	14.55	**12.54**	**14.96**	**1.18**	9.35	10.77	0.81	8.88	9.96	0.72	9.74	**8.87**
	2	14.17	26.56	6.33	17.54	**12.12**	**13.97**	**1.07**	9.41	10.8	0.81	**8.68**	10.16	1	9.76	8.87
	3	15.75	29.28	7.86	20.57	**13.5**	**17.25**	**2.58**	9.75	11.25	0.9	9.25	10.46	0.71	9.79	**9.21**
	4	14.9	28.48	7.93	18.09	**12.96**	**15.69**	**1.45**	9.55	11.03	0.89	**8.95**	10.35	0.85	9.31	9.44
	5	12.82	29.18	7.88	14.81	**12.68**	**14.36**	**1**	9.78	11.36	1.06	**9.09**	10.65	0.93	9.77	9.99
250 (15)	1	37.64	71.63	20.16	46.63	**17.7**	**34.51**	**6.2**	15.14	16.51	0.69	**14.04**	15.08	0.49	14.39	15.12
	2	28.9	74.73	19.46	30.76	**20.6**	**27.89**	**3.27**	15.2	16.33	0.67	**14.11**	14.99	0.48	14.24	14.6
	3	27.31	69.67	18.66	32.57	**22.18**	**28.36**	**3.09**	15.08	16.19	0.56	**13.8**	14.86	0.47	14.89	14.74
	4	29.42	75.44	19.86	32.97	**21.53**	**25.62**	**1.83**	15.49	16.77	0.62	**14.24**	15.38	0.48	14.68	15.19
	5	35.66	70.66	17.85	36.71	**21.63**	**29.46**	**3.7**	15.42	16.53	0.58	**14.11**	15.1	0.48	14.8	14.65
500 (20)	1	48.18	148.07	40.65	75.99	**27.73**	**50.84**	**13.53**	21.72	22.86	0.51	**19.39**	20.4	0.43	19.49	21.09
	2	60.15	146.37	40.38	63.69	**28.18**	**47.13**	**8.42**	21.46	22.52	0.46	**19.09**	20.17	0.42	19.75	20.89
	3	45.49	149.61	40.86	66.52	**33.92**	**47.88**	**8.9**	21.51	22.78	0.5	**19.42**	20.41	0.41	20.27	19.97
	4	63	148.34	40.22	68.69	**29.46**	**49.4**	**8.93**	21.82	22.85	0.47	**19.41**	20.46	0.46	19.58	21.88
	5	41.77	146.8	42.73	71.13	**28.09**	**50.97**	**13.2**	21.37	22.52	0.51	**18.86**	20.05	0.44	19.75	19.91
1000 (25)	1	90.01	321.07	84.9	158.4	**43.36**	**90.12**	**27.06**	30.97	32.19	0.41	**27.22**	28.26	0.43	29.53	29.75
	2	95.83	318.59	83.48	171.84	**41.54**	**90.73**	**33.04**	30.9	32.05	0.42	**27.08**	28.12	0.41	28.15	29.32
	3	94.02	312.7	85.72	150.01	**41.71**	**87.47**	**25.65**	30.69	31.77	0.42	**26.8**	27.83	0.4	28.85	29.03
	4	81.39	317.02	83.17	165.94	**44.75**	**90.59**	**28.04**	30.93	32.18	0.43	**27.05**	28.21	0.4	29.59	30.2
	5	70.55	318.52	81.37	167.49	**43.89**	**90.81**	**28.86**	30.85	31.93	0.42	**26.5**	27.91	0.42	27.68	28.88

Table 3: Results obtained on the Beasley Euclidean benchmarks, five instances each of 50, 100, 250, 500 and 1000 node graphs for the CBTC+HT, CBLSoC-lite, CBLSoC, RTC, RGH+HT, QCH-Greedy and QCH-LSoC heuristics

n=1000	CBTC		RTC		CdA		CdB				QCH-Greedy)			QCH-LSoC		
D	**Mean**	**SD**	**mean**	**SD**	**mean**	**SD**	**mean**	**SD**	t_{max}	**(s)**	**mean**	**SD**	t_{avg}	**mean**	**SD**	t_{avg}
(Even D)																
4	329.0261	6.02	146.4919	3.88	68.3241	0.72	**68.3226**	0.7	2.54	0.09	68.65	0.53	0.11	68.64	0.53	0.1547
6	306.2655	9.02	80.8636	2.4	47.4045	4.85	**47.1702**	4.61	4.55	0.49	54.17	0.57	0.0973	50.13	0.55	0.1247
8	288.3842	7.52	53.2535	1.33	37.0706	1.35	**36.9408**	1.34	5.92	0.42	46.3	0.65	0.088	42.94	0.53	0.108
10	266.3665	9.01	41.1201	0.68	33.546	0.67	**33.3408**	0.66	6.79	0.42	41.4	0.59	0.0933	39.28	0.26	0.104
12	250.0016	8.01	35.759	0.47	32.2571	0.48	**31.9561**	0.44	7.11	0.33	37.81	0.63	0.0767	36.68	0.4	0.1027
14	237.1403	6.28	33.3644	0.3	31.379	0.37	**31.0176**	0.33	7	0.64	35.36	0.47	0.0767	34.85	0.55	0.0987
16	224.3123	5.72	32.1965	0.24	30.7937	0.33	**30.4287**	0.29	7.2	0.72	33.22	0.42	0.074	33.4	0.67	0.0887
18	210.9872	7.63	31.5826	0.24	30.5182	0.29	**30.1348**	0.27	7.32	0.81	31.8	0.46	0.066	32.28	0.34	0.094
20	197.1772	7.99	31.2682	0.22	30.3116	0.31	30.0384	0.28	7.57	0.76	**30.41**	0.46	0.0647	31.27	0.43	0.084
22	183.0157	8.03	31.0864	0.22	30.2344	0.3	30.0739	0.28	8.56	0.98	**29.48**	0.35	0.0653	30.66	0.49	0.0747
24	172.8251	10.59	30.9921	0.23	30.0202	0.23	30.1603	0.27	8.28	1.41	**28.99**	0.53	0.066	29.92	0.55	0.072
(Odd D)																
5	241.3032	5.09	117.3238	2.22	62.2867	0.76	**62.0646**	0.67	24.59	2.02	68.08	0.43	0.1113	68.08	0.44	0.156
7	222.1441	4.5	67.7577	1.31	46.7291	3.92	**46.4112**	3.73	27.94	1.79	53.76	0.58	0.098	49.81	0.55	0.1267
9	204.6141	6	47.3168	0.85	37.0224	1.25	**36.8904**	1.27	18.27	1.68	46.04	0.67	0.0953	42.64	0.48	0.1093
11	189.7513	4.62	38.4754	0.5	33.414	0.7	**33.1749**	0.66	13.97	0.71	41.16	0.58	0.088	39.13	0.36	0.106
13	175.7382	4.23	34.5154	0.32	32.1094	0.43	**31.8041**	0.41	12.79	1.17	37.58	0.65	0.08	36.42	0.36	0.106
15	163.1926	4.31	32.7069	0.25	31.2654	0.35	**30.8941**	0.32	11.03	1.27	35.16	0.52	0.0767	34.68	0.53	0.0973
17	149.9852	5.14	31.8467	0.23	30.7699	0.33	**30.3664**	0.3	8.93	0.94	33.06	0.43	0.0727	33.22	0.64	0.09
19	139.973	4.32	31.4048	0.21	30.535	0.29	**30.0837**	0.27	7.91	1.08	31.67	0.49	0.0707	32.13	0.38	0.0853
21	128.183	4.9	31.1697	0.23	30.3017	0.3	**30.0384**	0.27	7.6	0.71	30.25	0.44	0.0647	31.12	0.34	0.0853
23	119.5551	4.46	31.0421	0.22	30.0627	0.24	30.1166	0.31	6.96	0.81	**29.34**	0.36	0.0653	30.59	0.52	0.08
25	110.6725	4.39	30.9772	0.23	29.945	0.21	30.1393	0.24	6.68	0.89	**28.87**	0.54	0.0713	29.78	0.52	0.0707

Table 4: Results obtained on the Beasley Euclidean benchmark instances: thirty independent runs on 1000 node graphs for CBTC, RTC, the Cluster-based Heuristic with two different cluster root assignment strategies (Cd^A and Cd^B), one run each of QCH-Greedy and QCH-LSoC heuristics

The QCH heuristics produce low cost BDSTs when the diameter bound is small, giving results that are competitive with RTC+HT. For example, on the results obtained for up to 1000 vertex graphs in table 3, the lower cost BDST of the two QCH Heuristics is no worse than 3.25% on average, with the QCH-Greedy and QCH-LSoC heuristics producing slightly lower cost BDSTs than RGH+HT on three instances. The QCH heuristics also do well when D is increased, cf. tables 1 and 2.

Gruber and Raidl [1] present the results obtained for their hierarchical clustering based heuristic on very small diameter bounds, averaged over all fifteen instances of 1000 vertex graphs from Beasley's Euclidean Steiner data set. Table 4 gives the results obtained by the proposed heuristics for the same set of diameter bounds and problem instances.

The maximum running times mentioned in the table for Gruber et al.'s heuristic are given in [1] and are as such not directly comparable with the mean computational time quoted for the proposed heuristics, as they were obtained on systems with different configurations.

The mean BDST costs given for CBTC, RTC and the two variants of the Hierarchical Clustering heuristic (Cd^A and Cd^B) were obtained over thirty independent runs on the fifteen 1000 vertex Euclidean graph instances. The parameters t_{max} and [s] represent the average and corresponding standard deviation (SD) over all instances, of the maximum running time of Cd^A and Cd^B.

The mean BDST costs and SD given for the two QCH heuristics represent the mean and SD obtained from a single run of the heuristics on each instance. Gruber et al. [1] also use a variable neighborhood descent strategy to further improve the results obtained by the hierarchical clustering-based heuristic; this is shown to work well only on low values of D (for instance when D is less than 14 on the 1000 vertex graphs).

The Hierarchical Clustering heuristic outperforms the CBTC and RTC heuristics by a wide margin; with increasing D, this margin is seen to narrow down. The CBLSoC heuristic returns higher cost BDSTs as compared to the RTC heuristic on small diameter bounds (tables 1 and 2), and is hence not tested in this case. The QCH heuristics, on the other hand, still give good results, performing much better than CBTC and RTC, and, as the value of D is increased further, outperforming Cd^B on the last five diameter bounds considered in the tests (table 4). Even on very tight diameter bounds, the QCH heuristics perform well: for example, the gap in solution quality for the smallest diameter bound considered in the test (D=4) is less than 0.5%.

6 Conclusions

The Euclidean Bounded Diameter Minimum Spanning Tree problem is to find a minimum spanning tree whose diameter does not exceed a specified number of edges, in the domain of graphs whose vertices are points in two dimensional space and edges are the Euclidean distances between vertices. The problem is known to be NP-hard, and hard to approximate, which motivates the search for effective heuristic strategies that are able to quickly find low cost BDSTs. This paper presents some simple fast and effective heuristic strategies and compares their performance with that of several extant heuristics for this problem over a wide range of benchmark problems, including a test suite of very large Euclidean dense graphs. One of the proposed heuristic approaches, called CBLSoC-lite, uses a less greedy node selection policy as compared to the OTTC and CBTC heuristics and builds low cost BDSTs in time that is atleast $O(n)$ faster than any of the extant heuristics. Running this heuristic starting from each graph node and returning the lowest cost BDST so obtained requires O(n^3) time but leads to better BDSTs. The other heuristic strategy starts with an empirically fixed tree "backbone" and appends the remaining nodes using either a greedy or CBLSoC-based node selection policy.

The heuristics presented in this work are classified in figure 2 into two categories, those that work on "standard" sized problems and those that are also able to solve large problem instances in reasonable time, and then ranked in increasing order of mean tree costs obtained as the diameter bounds go from small/tight to large/relaxed. Heuristics that perform competitively in a particular range share the same rank.

Problem Size	Diameter Bound	Heuristics in increasing order of mean tree costs
Standard	Small	1. Cd^B 2. Cd^A 3. RGH+HT/RTC/QCH-Greedy 4. QCH-LSoC 5. CBLSoC 6. CBLSoC-lite 7. CBTC+HT/CBTC 8. OTTC
	Large	1. CBLSoC 2. CBLSoC-lite/QCH-Greedy 3. QCH-LSoC 4. CBTC 5. OTTC 6. RTC
Large	Small	1. QCH-LSoC/RTC 2. QCH-Greedy 3. CBLSoC (limited range due to higher computation time) 4. CBLSoC-*lite*
	Large	1. QCH-Greedy 2. QCH-LSoC 3. CBLSoC-*lite*

Figure 2: *Performance-based ranking of the heuristics*

The OTTC heuristic produces spanning trees with larger costs, because it always uses low cost edges to build the tree, thus necessitating the later vertices to be appended to the tree through higher cost edges. As computational results show, this drawback is especially obvious when the diameter bound is small. The CBTC heuristic is faster and obtains lower cost BDSTs, but it also uses the same greedy premise as OTTC and hence suffers from the same drawbacks.

The CBLSoC heuristic is shown to perform better than the OTTC and CBTC heuristics on all of the benchmark instances used in this work. The CBLSoC-lite variant, which has a running time of O(n2), outperforms OTTC on every instance and produces lower mean costs vis-à-vis the CBTC heuristic on several instances. On problem instances with small diameter bounds, the randomized heuristic RTC outperforms the other extant heuristics, but this does not hold true when the diameter bound is increased. The improved RGH+HT and CBTC+HT heuristics are able to improve the solution quality as compared to RTC and CBTC, but they retain the drawbacks inherent in both these heuristics, thus rendering the RTC variant unsuitable for larger diameter bounds, and the CBTC variant unsuitable for low diameter bounds. The hierarchical clustering-based heuristic returns the lowest cost BDSTs on standard problems for very small diameter bounds, but its performance worsens with increasing diameter bound.

The proposed QCH heuristics compare favorably with RTC on low diameter bounds, and generally do better than all the other heuristics as the diameter constraint is relaxed. Furthermore, the lower running time requirements of the CBLSoC-lite heuristic and the QCH heuristics means that they can be used effectively for solving much larger problems than have been hitherto attempted.

Acknowledgements

The authors are extremely grateful to the Revered Chairman, Advisory Committee on Education, Dayalbagh for the continuous guidance in all their endeavors. The authors also thank Prof. Bryant A. Julstrom, St. Cloud State University, Minnesota, USA for providing the extended suite of problem instances for the BDMST problem.

References

[1] Gruber, M. and Raidl, G.R. (2009) Solving the Euclidean Bounded Diameter Minimum Spanning Tree Problem by Clustering-Based (Meta-) heuristics. *In: R. Moreno-Diaz et al., (Eds.), EUROCAST 2009, LNCS 5717*, Springer, pp. 665-672.

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

[3] Bala, K., Petropoulos, K., Stern, T.E. (1993) Multicasting in a linear lightwave network, *IEEE INFOCOM'93*, IEEE Press, pp. 1350-1358.

[4] Bookstein, A., Klein, S.T. (1996) Compression of correlated bit-vectors, *Information Systems*, Elsevier, Vol. 16(4), pp. 110-118.

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

[6] Kortsarz, G., Peleg, D. (1997) Approximating shallow-light trees, *Eighth ACM-SIAM Symposium on Discrete Algorithms*, pp. 103-110.

[7] Achuthan N.R. and Caccetta L. (1992) Minimum weight spanning trees with bounded diameter, *Australasian Journal of Combinatorics*, University of Queensland Press, Vol. 5, pp. 261–276.

[8] Achuthan N. R., Caccetta L., Caccetta P., and Geelen J. F. (1994) Computational methods for the diameter restricted minimum weight spanning tree problem, *Australasian Journal of Combinatorics*, University of Queensland Press, vol. 10, pp. 51-71.

[9] Gouveia L. and Magnanti T.L. (2003) Network flow models for designing diameter constrained minimum spanning and Steiner trees, *Networks*, Wiley, Vol. 41, no. 3, pp. 159–173.

[10] Deo N., and Abdalla A. (2000) Computing a Diameter-Constrained Minimum Spanning Tree in Parallel, *Italian Conference on Algorithms and Complexity*, CIAC-2000, Springer, LNCS1767, pp. 17–31.

[11] Julstrom B.A. (2009) Greedy heuristics for the bounded diameter minimum spanning tree problem, *Journal of Experimental Algorithmics*, ACM, vol. 14, no. 1, pp. 1-14, February 2009.

[12] Singh A. and Gupta A.K. (2007) Improved heuristics for the bounded-diameter minimum spanning tree problem, *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, Springer, vol. 11, no. 10, pp. 911–921.

[13] Singh A. and Saxena R. (2009) Solving bounded diameter minimum spanning tree problem with improved heuristics, *ADCOM 2009*, pp. 90-95.

[14] Gruber M. and Raidl G.R. (2009) Exploiting hierarchical clustering for finding bounded diameter minimum spanning trees on Euclidean instances, *Genetic and Evolutionary Computation Conference*, Springer, Montreal, Canada, pp. 263-270.

[15] Patvardhan C. and Prakash V. P. (2009) Novel Deterministic Heuristics for Building Minimum Spanning Trees with Constrained Diameter, *Pattern Recognition and Machine Intelligence*, Springer-Verlag, LNCS 5909, pp. 68-73, December 2009.

[16] Patvardhan, C.; Prakash, V.P.; Srivastav, A. (2014) Parallel heuristics for the bounded diameter minimum spanning tree problem, *India Conference (INDICON), 2014 Annual IEEE*, IEEE Press, pp.1-5, 11-13 Dec. 2014. (doi: 10.1109/INDICON.2014.7030575)

[17] Handler, G. Y. (1978) Minimax location of a facility in an undirected graph, *Transportation Science*, INFORMS, pp. 287–293, 7, 1978.

[18] Gruber M., Van Hemert J., and Raidl G. R. (2006) Neighborhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS,

EA, and ACO, *M. Keijzer et al., Eds., Proceedings of Genetic and Evolutionary Computation Conference 2006*, Springer-Verlag, vol. 2, pp. 1187-1194.

[19] Prim, R.C. (year) Shortest connection networks and some generalizations, *Bell System Technical Journal*, vol. 36, pp. 1389-1401.

[20] Raidl G.R. and Julstrom B.A. (2003) Greedy heuristics and an evolutionary algorithm for the bounded diameter minimum spanning tree problem, *ACM Symposium on Applied Computing*, ACM Press, pp. 747-752.

[21] Julstrom B.A. and Raidl G.R. (2003) A permutation-coded evolutionary algorithm for the bounded-diameter minimum spanning tree problem, *Genetic and Evolutionary Computation Conference's Workshops Proceedings, Workshop on Analysis and Design of Representations*, pp. 2–7.

[22] Binh H. and Nghia N. (2009) New multiparent recombination in genetic algorithm for solving bounded diameter minimum spanning tree problem, *First Asian Conference on Intelligent Information and Database Systems*, pp. 283-288.

[23] Javad Akbari Torkestani (2012) An adaptive heuristic for the bounded diameter minimum spanning tree problem, *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, Springer, vol. 16, no. 11, pp. 1977-1988.