Parallel Heuristics for the Bounded Diameter Minimum Spanning Tree Problem

C. Patvardhan and V. Prem Prakash
Department of Electrical Engineering
Dayalbagh Educational Institute, Agra
Agra – 282005, India
cpatvardhan@ieee.org, vpremprakash@acm.org

A. Srivastav
Institut für Informatik,
Christian-Albrechts-Universität zu Kiel,
24098 Kiel, Germany
E-mail: asr@informatik.uni-kiel.de

***Abstract* — Given a connected, undirected graph G = (V, E) on n = |V| vertices, an integer diameter bound D ≥ 2 and non-zero edge weights associated with each edge e ∈ E, a bounded diameter minimum spanning tree (BDMST) on G is defined as a spanning tree T ⊆ E on G of minimum edge cost, w(T) = Σ $_{\forall e \in T}$ w(e) and tree diameter no greater than D. The problem of computing BDMSTs is known to be NP-Hard for 4 ≤ D < n-1, and hard to approximate. Well known greedy and randomized greedy heuristic strategies are extant in the literature which build low cost diameter-constrained spanning trees in O(n^3) time. The greedy heuristics perform better on graphs with random edge weights, whereas the randomized greedy heuristics produce lower cost BDSTs on Euclidean graphs. Another recent heuristic uses a "less greedy" approach and performs competitively vis-à-vis the other heuristics. This paper presents parallel versions of some of these heuristics (the Center-based Tree Construction (CBTC), Randomized Tree Construction (RTC) and Center-based Least Sum-of-costs (CBLSoC) heuristics) implemented using OpenMP. The reason for choosing these heuristics for parallelization is that they have been shown to perform well over a wide variety of problem instances, whereas other extant heuristics in the literature are known to give perform well only on specific types of graphs. The performance of all the heuristics is compared on a wide range of standard benchmark instances comprising of completely connected Euclidean and random graphs.**

Keywords— Bounded diameter; minimum spanning tree; heuristics; parallel; OpenMP

I. INTRODUCTION

Given a connected, weighted, undirected graph G and a positive integer D, a bounded-diameter spanning tree (BDST) is a spanning tree T ⊆ E on G = (V, E), whose diameter is no greater than D. The Bounded-Diameter Minimum Spanning Tree (BDMST) problem seeks a BDST that minimizes the cost w(T) = $\Sigma_{\forall e \in T}$ w(e). The problem finds application in many domains, including distributed mutual exclusion [1], wire-based communication network design [2] and data compression for information retrieval [3]. 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 [4]. Furthermore, the problem is also known to be hard to approximate; it has been shown that no PTAS can be guaranteed to find a solution within log (n) of the optimum [5].

An exact algorithm for the BDMST Problem is given by Achuthan and Caccetta [6] and improved by Achutan et al. in [7]. Gouveia and Magnanti [8], 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 restricts their application to only very small problem instances; this is a key motivation for finding fast heuristics that can approximate low cost BDSTs on much larger problem sizes within reasonable time.

Ayman Abdalla and Narsing Deo [9] give a Prim's [10] 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). This heuristic consistently produces low cost BDSTs on graphs whose edge weights are randomly chosen. 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 heuristic is especially effective on Euclidean problem instances when the diameter bound D is small, but loses out to CBTC on other types of problem instances.

The *Center-based Least Sum-of-Costs* (CBLSoC) heuristic given by Patvardhan et al. [14] 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 on Euclidean problems. However, the performance of this heuristic on non-Euclidean graphs has not been reported thus far.

Other heuristics for the problem include a hierarchical clustering-based heuristic for the Euclidean variant of the BDMST problem given by Gruber and Raidl [13], which obtains low cost BDSTs on Euclidean graphs with very small diameter constraints, and improved variants of the CBTC and RTC heuristics [12]. Two other heuristics that work well only on Euclidean graphs are given in [14] which first try to construct an effective backbone comprising of a small, empirically chosen number of nodes at low depth which are appended to the tree via relatively longer edges, and then append the remaining graph nodes to the BDST.

The paper presents parallel versions of the CBTC, RTC and CBLSoC heuristics implemented using OpenMP. These have been selected for parallelization as they have been shown to perform well over a wider variety of graphs in contrast to the other abovementioned heuristics that give good performance only on specific types of graphs. The performance of the heuristics and speedup gained is studied using a large number of Euclidean and non-Euclidean benchmark graphs widely used in the literature.

Subsequent sections of the paper are organized as follows: section 2 describes the CBTC, RTC and CBLSoC heuristics, section 3 presents the computational results obtained on benchmark problems, and section 4 concludes and summarizes the paper.

II. HEURISTICS

A. *Center-based Tree Construction (CBTC) Heuristic*

The Center-Based Tree Construction (CBTC) heuristic [11] improves builds a BDST from the tree's center, keeping track of the depth of each node in the tree and ensuring that node depth does not exceed $\lfloor D/2 \rfloor$. The center of the BDST comprises of either one vertex (in case D is even) or two vertices connected by an edge (if D is odd). The heuristic returns the lowest cost BDST obtained over n runs, each starting from a different graph vertex, thereby requiring a total running time of $O(n^3)$. Pseudo-code for the CBTC heuristic is given in figure 1.

```
U[.] ← set of unconnected graph vertices
C[.] ← set of tree nodes with depth less than ⌊D/2⌋
for each vertex v0 ∈ V do                                          ...n
                                                               iterations
1. Set v0 as root
   U ← U − {v0}                                               ...O(1) time
   C ← {v0}
   depth[v0] ← 0
2. if D is odd, choose vertex v1 ∈ U|{cost(v0,v1) is minimal}
   U ← U − {v1}                                               ...O(1) time
   C ← C ∪ {v1}
   depth[v1] ← 0
   T ← T ∪ (v0, v1)
3. while U[.] not empty do
        find v ∈ U and u ∈ C such that cost(u, v) is minimal     ...O(n2)
        ∀u ∈ C, ∀v ∈ U                                             time
        T ← T ∪ (u, v)
        U ← U − {v}
        depth[v] ← depth[u] + 1
        if (depth[v] < ⌊D/2⌋)
            C ← C ∪ {v}
  end while
Return the tree with the lowest cost out of all the trees generated
above
```

Figure 1. Pseudo code for the CBTC Heuristic

B. *Randomized Tree Construction (RTC) Heuristic*

In the Randomized Tree Construction Heuristic (RTC), the center of the spanning tree comprises of one vertex (if D is even) or two connected vertices (if D is odd) randomly selected from the set of graph nodes. This is similar to the approach taken by the CBTC heuristic. 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 figure 2.

```
U ← set of unconnected graph vertices
C ← set of tree nodes with depth less than ⌊D/2⌋
repeat n times                                                ...n iterations
1. Randomly choose a vertex v0 ∈ U and set as root               ...O(1) time
   U ← U − {v0}
   C ← {v0}
   depth[v0] ← 0
2. if D is odd, randomly choose another vertex v1 ∈ U            ...O(1) time
   U ← U − {v1}
   C ← C ∪ {v1}
   depth[v1] ← 0
   T ← T ∪ (v0, v1)
3. while U[.] not empty do                                      ...O(n²) time
   choose a random vertex v ∈ U
   find vertex u ∈ C such that cost(u, v) is minimal ∀u ∈ C
   T ← T ∪ (u, v)
   U ← U − {v}
   depth[v] ← depth[u] + 1
   if (depth[v] < ⌊D/2⌋)
      C ← C ∪ {v}
   end while
Return the tree with the lowest cost out of all the trees generated
above
```

Figure 2. Pseudo code for the RTC Heuristic

C. *The CBLSoC Heuristic*

The Center-based Least Sum-of-Costs (CBLSoC) heuristic iterates through the graph nodes, taking each vertex as the root node. If the diameter bound is odd, then the graph vertex with lowest mean cost to all other graph vertices is selected as a second root node and the edge between the two nodes is added to the BDST. Each next graph vertex chosen to be appended to the BDST is the vertex with the lowest mean cost to all the remaining graph vertices: this vertex is appended to the tree greedily via the lowest cost edge that does not violate the diameter bound.

```
A ← Adjacency matrix containing edge weights of the graph G
U ← Set of unconnected graph vertices

for each vertex u in G do                                                   ...O(n) time
1. T ← {u}; U ← U \ {u}                                                     ...O(1) time
2. If D is odd
       choose v ∈ V \ {u} such that sum of entries in row v of A is minimal   ...O(n) time
3. T ← {v}; U ← U \ {v}                                                     ...O(1) time
4. while U[.] not empty do                                                  ...O(n) time
       choose x ∈ V \ T such that $\sum_{y \in V\setminus T, y \neq x} cost(x,y)$ is minimal and
       diameter constraint is not violated                                  ...O(n) time
       T ← T ∪ {x} via the lowest cost edge to a vertex in T;               ...O(n) time
       U ← U \ {x}                                                          ...O(n) time
Return the tree with the lowest cost out of all the trees generated above
```

Figure 3. Pseudo code for the CBLSoC Heuristic

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 $\lfloor D/2 \rfloor$. 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 each iteration the heuristic. The CBLSoC heuristic iteratively builds BDSTs starting once from each graph vertex and returns the lowest cost BDST thus obtained. This requires a total running time of $O(n^3)$ for the heuristic. A simple post-processing step tries to reappend each node to the BDST using a lower cost edge if possible, without violating the diameter bound. This step requires at most $O(n^2)$ time and does not affect the overall time complexity of the heuristic. Pseudo-code for CBLSoC is given in figure 3.

III. COMPUTATIONAL RESULTS AND DISCUSSION

The Euclidean Steiner Problem data sets given in Beasley's OR-Library[1] have been used extensively in the literature for benchmarking 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. Thirty complete graphs of n=100, 250, 500 and 1,000 vertices with edge weights randomly chosen from the interval [0.01, 0.99] were also generated by Julstrom [9]. The diameter bounds in all the tests were always less than the smallest diameter of an unconstrained MST on each set of graphs.

All the heuristics were coded in the C programming language and parallel versions of the CBTC, RTC and CBLSoC heuristics implemented using OpenMP on a Dell Precision T-5500 Workstation with 2 Intel Xeon 2.4-Gigahertz processors having 6-cores each and 11 GB RAM running Red Hat Enterprise Linux 6.

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

The heuristics are compared in terms of mean BDST costs obtained ($\underline{X}$), standard deviation (SD), computation time ($\underline{t}$) and speedup (for the parallel implementations) on Euclidean problem instances in table 1 and random graphs in table 2.

The mean BDST costs for the CBTC, RTC and CBLSoC heuristics given in table 1 show that the CBLSoC heuristic produces lower mean costs vis-à-vis the CBTC heuristic on all Euclidean instances. The RTC heuristic produces relatively lower cost trees, but only when the diameter bound is very small; as the diameter bound is increased the lowest cost BDSTs are always returned by CBLSoC.

To understand this behavior, we observe that the OTTC and CBTC heuristics always greedily append to the tree, the feasible graph vertex with the lowest cost (in the Euclidean context, the shortest edge) to the tree. So 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 chooses the next graph vertex randomly, and thus might have a 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, the RTC heuristic usually returns trees with lower costs than any of the other heuristics. 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. As can be seen from table 1, the heuristic produces barely any improvements in BDST costs in several cases and is eventually surpassed in performance by the other heuristics as the diameter bound is increased further.

The CBLSoC 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 BDST. This heuristic consistently obtains lower cost BDSTs in comparison to the CBTC heuristic, and does better than the RTC heuristic when the diameter bound is large.

The results obtained on the non-Euclidean (random edge-weight) instances for the CBTC, RTC and CBLSoC heuristics are given in table 2. On these instances, the RTC heuristic performs poorly, always producing mean BDST costs that are much worse than those obtained by the CBTC and CBLSoC variants. The CBTC and CBLSoC heuristics produce better (lower cost) BDSTs on small instances (up to n=250) with tight diameter bounds, however as the instance size increases (n=500, 1000) it is seen that the CBTC heuristics produce better cost trees than the CBLSoC heuristic does.

This might be explained as follows: in the non-Euclidean scenario, each vertex shares a few low cost edges with some other vertices. The CBTC and CBLSoC heuristics tend to greedily choose those edges to the tree, whereas RTC always chooses its vertices randomly, as often as not making a costly choice in the process.

Table 1. Results obtained on Euclidean benchmark instances, thirty each of 100, 250, 500 and 1000 node graphs for the CBTC, RTC and CBLSoC heuristics

Instances		CBTC			$CBTC_{OpenMP}$		RTC			RTC_{OpenMP}		CBLSoC			$CBLSoC_{OpenMP}$	
N	D	$\underline{X}$	SD	$\underline{t}$	t_{par}	Speedup	$\underline{X}$	SD	$\underline{t}$	t_{par}	Speedup	$\underline{X}$	SD	$\underline{t}$	t_{par}	Speedup
100	5	26.48	1.51	0.004	0.0006	6.67	**15.39**	0.66	0.0023	0.0031	0.73	22.33	1.43	0.013	0.0014	9.57
	10	15.59	1.28	0.0047	0.0006	7.58	**9.76**	0.29	0.0042	0.0034	1.23	12.54	0.96	0.011	0.0015	7.53
	15	10.95	1.00	0.0054	0.0007	7.71	9.23	0.27	0.0059	0.0034	1.71	**9.15**	0.57	0.011	0.0021	5.29
	25	7.69	0.34	0.0068	0.0008	7.93	9.16	0.25	0.006	0.0033	1.81	**7.32**	0.24	0.011	0.0015	7.27
250	10	49.07	2.99	0.0701	0.0070	10	**16.84**	0.31	0.0531	0.0180	2.95	30.91	1.80	0.167	0.0175	9.53
	15	36.44	3.15	0.0747	0.0077	9.74	**15.32**	0.22	0.0715	0.0206	3.48	21.79	1.79	0.197	0.0203	9.70
	20	26.17	2.36	0.082	0.0084	9.8	**15.02**	0.22	0.0842	0.0219	3.85	17.84	1.13	0.165	0.018	9.16
	40	12.51	0.83	0.1085	0.0108	10.05	14.98	0.23	0.0856	0.0210	4.07	**11.47**	0.20	0.159	0.0166	9.58
500	15	94.38	5.56	0.5597	0.0552	10.14	**22.21**	0.33	0.5688	0.0893	6.37	44.39	3.09	1.447	0.146	9.91
	30	46.51	4.13	0.6682	0.0641	10.42	**21.42**	0.34	0.6792	0.0951	7.14	25.33	2.01	1.655	0.1449	11.42
	45	25.63	2.18	0.7603	0.0769	9.89	21.45	0.36	0.6819	0.0930	7.34	**17.80**	0.78	1.401	0.1508	9.29
	60	17.72	0.92	0.8909	0.0839	10.61	21.42	0.34	0.6762	0.0914	7.4	**16.03**	0.29	1.595	0.1383	11.53
1000	20	195.96	7.97	4.5729	0.4251	10.76	**31.20**	0.24	5.0617	0.5232	9.68	51.65	1.70	11.158	1.0824	10.31
	40	99.57	7.86	5.1829	0.4735	10.95	**30.81**	0.26	5.3586	0.5497	9.75	33.22	2.93	11.58	1.2539	9.24
	60	50.97	5.24	5.9103	0.5478	10.79	30.81	0.26	5.3863	0.5483	9.82	**27.71**	1.68	12.012	1.1288	10.64
	100	23.41	0.78	7.4194	0.6784	10.94	30.81	0.26	6.4516	0.5466	11.8	**22.57**	0.19	14.019	1.3211	10.61

Table 2. Results obtained on random problem instances, thirty each of 100, 250, 500 and 1000 node graphs for the CBTC, RTC and CBLSoC heuristics

Instances		CBTC			$CBTC_{OpenMP}$		RTC			RTC_{OpenMP}		CBLSoC			$CBLSoC_{OpenMP}$	
N	D	$\underline{X}$	SD	$\underline{T}$	t_{par}	Speedup	$\underline{X}$	SD	$\underline{t}$	t_{par}	Speedup	$\underline{X}$	SD	$\underline{t}$	t_{par}	Speedup
100	5	5.35	0.33	0.01	0.0036	2.75	6.99	0.37	0.01	0.0031	3.19	**5.23**	0.38	0.015	0.004	3.41
	7	**3.34**	0.19	0.01	0.0037	2.72	4.75	0.25	0.01	0.003	3.28	3.77	0.23	0.015	0.004	3.48
	10	**2.56**	0.13	0.01	0.0038	2.66	4.05	0.26	0.01	0.0031	3.20	3.22	0.20	0.015	0.004	3.49
	15	**2.26**	0.11	0.01	0.0038	2.62	3.86	0.20	0.01	0.0033	2.99	3.30	0.24	0.016	0.004	3.65
250	5	10.14	0.43	0.11	0.0265	4.15	12.91	0.52	0.08	0.0179	4.48	**9.40**	0.36	0.213	0.037	5.80
	10	**4.45**	0.16	0.13	0.0279	4.66	6.62	0.23	0.1	0.0206	4.85	5.21	0.22	0.231	0.036	6.41
	15	**3.99**	0.13	0.15	0.0293	5.11	6.13	0.21	0.12	0.0198	6.05	5.19	0.22	0.232	0.037	6.26
	20	**3.80**	0.12	0.15	0.0299	5.01	6.08	0.23	0.13	0.0251	5.18	5.01	0.23	0.23	0.036	6.42
500	10	7.25	0.11	1.04	0.1469	7.08	10.02	0.20	0.75	0.0779	9.63	8.07	0.15	1.761	0.214	8.24
	15	**6.61**	0.08	1.10	0.1551	7.09	9.26	0.12	0.88	0.0904	9.74	8.03	0.19	1.911	0.224	8.53
	20	**6.36**	0.06	1.20	0.1576	7.61	9.14	0.20	0.95	0.0948	10.02	7.73	0.16	1.804	0.219	8.25
	30	**6.21**	0.06	1.29	0.1638	7.88	9.12	0.20	0.96	0.0964	9.96	7.72	0.16	1.789	0.215	8.31
1000	10	12.71	0.10	10.95	0.9749	11.23	16.09	0.21	8.38	0.7493	11.18	13.52	0.17	16.906	1.479	11.43
	20	**11.50**	0.05	12.16	1.0286	11.82	14.69	0.16	10.17	0.8517	11.94	12.96	0.19	18.012	1.514	11.90
	30	**11.30**	0.04	13.50	1.1594	11.64	14.75	0.16	10.5	0.8894	11.81	12.95	0.19	18.097	1.543	11.73
	50	**11.19**	0.03	13.61	1.1959	11.38	14.69	0.16	10.49	0.8969	11.70	12.95	0.19	17.381	1.528	11.37

This could account for the poor performance of RTC vis-à-vis the two deterministic heuristics. As the diameter bound is increased in each instance, the CBLSoC heuristic consistently performs better than RTC and but loses out to the CBTC heuristic.

Parallel versions of the three heuristics were developed using OpenMP in a multi-core environment. In tables 1 and 2, the mean computation time (t_{par}) and speedup obtained are given for each heuristic. For the RTC heuristics, the speedup was not significant on small Euclidean instances, but improved quickly with problem size. However, for the CBTC and CBLSoC heuristics, considerable speedup was obtained on small problem sizes also. For instance, the average speedup obtained on 100 vertex Euclidean graphs was 1.37, as compared to about 7.4 for CBTC and CBLSoC heuristics. On non-Euclidean instances, the speedup was considerably higher for the 500 and 1000 vertex instances for all the three heuristics.

IV. CONCLUSIONS

Given a connected, weighted, undirected graph G and a positive integer D, the Bounded-Diameter Minimum Spanning Tree problem finds a lowest cost spanning tree that only contains paths with at most D edges. Two well known greedy heuristics for this problem are the Center-based Tree Construction (CBTC) heuristic and the Randomized Tree Construction heuristic (RTC), both of which run in $O(n^3)$ time. A relatively "less greedy" approach is followed by the Center-based Least Sum-Of-Costs heuristic (CBLSoC) which also has a running time of $O(n^3)$. The CBTC heuristic obtains the best BDSTs on most non-Euclidean problem instances but loses out to the CBLSoC and RTC heuristics on Euclidean graphs. The RTC heuristic obtains the lowest cost BDSTs on Euclidean instances, but only for small diameter bounds. Moreover, it performs poorly in comparison to the other two heuristics on the non-Euclidean benchmarks. The CBLSoC heuristic, on the other hand, performs competitively on both types of problem instances. The heuristic consistently outperforms the CBTC heuristic on Euclidean graphs and obtains the lowest cost BDSTs on large diameter bounds. On non-Euclidean benchmarks also, the heuristic always returns lower cost BDSTs than the RTC heuristic, and obtains the lowest cost BDST on some problem instances. Parallel versions of all three heuristics are implemented using OpenMP, a widely used directive-based, fork-join model for shared memory parallelism on a realistic number of processors. The effectiveness of the parallel versions is seen from the consistently good speedup obtained by the parallel heuristics, and motivates their use to quickly solve larger problem instances.

Acknowledgments

The authors are extremely grateful to the Revered Chairman, Advisory Committee on Education, Dayalbagh for the continued 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] Raymond, K.: A tree-based algorithm for distributed mutual exclusion. ACM Transactions on Computer Systems, vol. 7 (1), pp. 61-77 (1989).

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

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

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

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

[6] Achuthan N.R. and Caccetta L., "Minimum weight spanning trees with bounded diameter," *Australasian Journal of Combinatorics*, vol. 5, pp. 261–276, 1992.

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

[8] Gouveia L. and Magnanti T.L., "Network flow models for designing diameter constrained minimum spanning and Steiner trees," *Networks*, vol. 41, no. 3, pp. 159–173, 2003.

[9] Deo N., and Abdalla A., "Computing a Diameter-Constrained Minimum Spanning Tree in Parallel," Italian Conference on Algorithms and Complexity (CIAC 2000), LNCS 1767, pp. 17–31, 2000.

[10] Prim, R.C.: Shortest connection networks and some generalizations. Bell System Technical Journal, vol. 36, pp. 1389-1401 (1957).

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

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

[13] Gruber M. and Raidl G.R., "Exploiting hierarchical clustering for finding bounded diameter minimum spanning trees on Euclidean instances", GECCO'09, July 8–12, 2009, Montréal, Québec, Canada.

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