Novel Deterministic Heuristics for Building Minimum Spanning Trees with Constrained Diameter

C. Patvardhan[1] and V. Prem Prakash[2]

[1] Faculty of Engineering, Dayalbagh Educational Institute, Agra, India
[2] Department of Information Technology, Anand Engineering College, Agra, India
cpatvardhan@ieee.org, vpremprakash@acm.org

Abstract. Given a connected, weighted, undirected graph G with n vertices and a positive integer bound D, the problem of computing the lowest cost spanning tree from amongst all spanning trees of the graph containing paths with at most D edges is known to be NP-Hard for $4 \leq D < n-1$. This is termed as the Diameter Constrained, or Bounded Diameter Minimum Spanning Tree Problem (BDMST). A well known greedy heuristic for this problem is based on Prim's algorithm, and computes BDMSTs in $O(n^4)$ time. A modified version of this heuristic using a tree-center based approach runs an order faster. A greedy randomized heuristic for the problem runs in $O(n^3)$ time and produces better (lower cost) spanning trees on Euclidean benchmark problem instances when the diameter bound is small. This paper presents two novel heuristics that compute low cost diameter-constrained spanning trees in $O(n^3)$ and $O(n^2)$ time respectively. The performance of these heuristics vis-à-vis the extant heuristics is shown to be better on a wide range of Euclidean benchmark instances used in the literature for the BDMST Problem.

Keywords: Heuristics, Diameter Constrained Minimum Spanning Tree, Bounded Diameter, Greedy.

1 Introduction

Given a connected, undirected graph G = (V, E) on |V| vertices, an integer bound $D \geq 2$ and non-zero edge costs associated with each edge $e \in E$, a diameter constrained, or bounded diameter spanning tree (BDST) is defined as a spanning tree $T \in E$ on G with tree diameter no greater than D. The BDMST Problem finds a bounded diameter spanning tree of minimum edge cost, $w(T) = \Sigma_{\forall e \in T} w(e)$. 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]. The problem finds application in several domains, including distributed mutual exclusion [4], wire-based communication network design [2], ad-hoc wireless networks and data compression for information retrieval [3].

S. Chaudhury et al. (Eds.): PReMI 2009, LNCS 5909, pp. 68–73, 2009.

Three branch-and-bound exact algorithms for solving the BDMST problem are given in [10], however, the exponential time complexity of these algorithms severely restricts their application to all but very small instances. Two greedy heuristics are proposed by Abdalla et al. [1], one of which starts by computing an unconstrained MST and then iteratively transforms it into a BDMST. This heuristic is computationally expensive and does not always give a BDMST for small values of n. The other heuristic is known as One-Time Tree Construction (OTTC), which modifies Prim's algorithm [8] for unconstrained MSTs to iteratively select the lowest cost non-tree vertex whose inclusion in the tree does not violate the diameter constraint. When run starting once from each node (and returning the lowest cost BDST thus obtained), the OTTC algorithm requires $O(n^4)$ running time. A faster center-based variant of OTTC, known as Center-Based Tree Construction (CBTC) and a better performing Randomized Tree Construction heuristic (RTC) for the BDMST problem are proposed in [7]. Center-Based Tree Construction grows a 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 improves the running time of OTTC by O(n). RTC is also a center-based heuristic based on Prim's algorithm, but with the proviso that the next node to be added to the MST is chosen at random and attached greedily. Greedily appending a randomly chosen node to the tree can be performed in linear time; building the BDST would require $O(n^2)$ time. This process is repeated n times and the lowest cost tree obtained is returned; thus the total running time incurred by the RTC heuristic is $O(n^3)$. The greedy heuristics given so far are extended by Singh and Gupta [11] with an $O(n^2)$ time post-processing step that re-attaches each node to the BDMST at a lower cost if possible. The performance of the OTTC, CBTC and RTC heuristics is presented on a wide range of benchmark problems in a recent work by Julstrom [9].

This paper presents a novel heuristic called the Least Sum-of-Costs Node Next (LSoC) heuristic that constructs the MST by greedily including the next node with the lowest mean cost to the remaining non-tree nodes. This heuristic requires $O(n^3)$ time, or totally $O(n^4)$ time when run starting from each node, and produces lower cost trees as compared to OTTC. An improved center-based version of the LSoC heuristic that requires $O(n^3)$ time and two variants of a faster heuristic designed specifically for Euclidean cases of the problem are also given. The performance of the proposed heuristics is shown to be better than existing heuristics on a wide range of Euclidean problem instances used in the literature for the BDMST problem. The rest of this paper is organized as follows: section 2 introduces the LSoC heuristic and presents the improved version of the LSoC heuristic and two fast heuristics that work well on Euclidean problems; section 3 discusses the results obtained on Euclidean benchmark instances and compares them with those obtained by the extant heuristics, and some conclusions are presented in section 4.

2 Proposed Heuristics

The Least Sum-of-Costs Node Next (LSoC) heuristic always selects as the next tree node, the non-tree vertex with the lowest total cost to the remaining non-tree vertices. This vertex is then appended to the tree via the lowest cost edge that does not violate

the diameter bound. Diameter checking is essentially an $O(n^2)$ breadth-first tree traversal that computes the maximum eccentricity between any two tree nodes; this check is performed n-1 times, resulting in a total running time of $O(n^3)$. In order to obtain a low cost BDST, the heuristic is run starting from each graph vertex. This adds a factor of n to the running time, but gives consistently better trees.

A center-based variant of LSoC (abbreviated to CBLSoC) starts with each node as center when D is even or each node plus the node with lowest mean cost to the remaining non-tree nodes as centers when D is odd. Thereafter the next node to be added to the tree is simply the node with the lowest mean cost to the remaining non-tree nodes, and is appended greedily to the tree, while ensuring that no tree node had depth greater than $\lfloor D/2 \rfloor$. Appending a node to the tree does not affect the depth of any tree node. Choosing the next node to append to the tree, and selecting its parent are both linear time operations, and are repeated n-1 times. Thus each BDST is constructed in $O(n^2)$ time; since this process is repeated starting from each of the n nodes, the total computation time required for CBLSoC is $O(n^3)$.

As mentioned already, OTTC (and CBTC) greedily appends nodes to the tree, with the result that the edges that form the backbone of the growing tree are typically short edges. Thus (possibly several of) the remaining nodes have to be appended using long edges, which add to the total cost of the tree. The relatively less greedy LSoC heuristic mitigates this to some extent, as the results in section 3 show. Another approach to this problem, designed specifically for Euclidean benchmark problems, tries to overcome this problem by setting up the tree center and some of the initial tree nodes empirically, in effect building a backbone comprising of a small number of nodes appended to the tree via relatively longer edges, and then constructing the rest of the BDST either greedily or using the LSoC heuristic. These heuristics, termed as Nine Quadrant Centers-based Heuristics (NQC), start by choosing the node(s) closest to the center of the (Euclidean) data plane as the tree center, and appending to the tree, the node closest to each of the eight respective quadrant centers of a uniform 3x3 matrix of the unit square representing the plane of data points, with the exception of the central quadrant (which in any case already contains the tree center). The node closest to the center of the unit square of Euclidean points is initially fixed as the center if (the diameter) D is even. On the other hand, if D is odd, then the next closest node is selected and added to the tree via the lowest cost edge, and the center of the spanning tree is formed using both these nodes. For each of the nine quadrants of the data plane, the graph vertex closest to a quadrant center is appended to the tree center and designated as a node of depth 1. The remaining vertices are now appended to the tree in a greedy manner (called the NQC-Greedy heuristic), or using the LSoC heuristic in selecting each next vertex to append to the tree (termed as the NQC-LSoC heuristic). Setting up the backbone of the BDST in this manner requires constant time in both NQC variants. Appending each node to the BDST requires $O(n^2)$ time in the greedy variant and linear time in the LSoC variant; thus the total running time for the NQC-Greedy heuristic works out to $O(n^3)$, whereas in the case of NQC-LSoC it amounts to $O(n^2)$. All three heuristics incorporate the Singh et al [11] post processing step to further improve the cost, which does not change the overall time complexity, and improves tree cost in several cases.

3 Performance on Benchmark Problems

Problem instances from the Euclidean Steiner Problem data set available in Beasley's OR-Library[1] have been predominantly used in the literature to benchmark the performance of heuristics and algorithms for the BDMST Problem. In a recent work, Julstrom [9] used a larger set of benchmark problems by combining the Beasley data sets with randomly generated Euclidean graphs, fifteen each of 100, 250, 500 and 1000 node instances, whose edge weights are the Euclidean distances between (randomly generated) points in the unit square. The heuristics presented in section 2 were implemented in C on a PIV 3-GHz processor with 2 GB RAM running Fedora 9, and tested on thirty instances of 100, 250, 500 and 1000 node graphs from this augmented test suite, totaling 120 graphs, and the mean ($\underline{X}$) and standard deviation (SD) of tree costs, and mean CPU times were obtained for each node size. The mean CPU times ($\underline{t}$) given in tables 1 and 2 for the existing and proposed heuristics were obtained on computing systems with different configurations, and are hence not directly comparable. However, the lesser computational complexity of the proposed algorithms is very clearly reflected in the times shown. The CBLSoC heuristic is compared with OTTC and CBTC in table 1.

Table 1. Results for the OTTC, CBTC and CBLSoC Heuristics on up to 1000 node graphs

Instances		OTTC			CBTC			CBLSoC		
n	D	$\underline{X}$	SD	$\underline{t}$	$\underline{X}$	SD	$\underline{t}$	$\underline{X}$	SD	$\underline{t}$
100	5	29.38	1.71	0.07	26.48	1.51	0.01	**22.34**	1.43	0.05
	10	18.43	1.86	0.07	15.59	1.28	0.01	**12.53**	0.97	0.04
	15	12.84	1.47	0.07	10.95	1.00	0.01	**9.15**	0.56	0.05
	25	8.06	0.59	0.07	7.69	0.34	0.02	**7.32**	0.24	0.04
250	10	58.20	5.38	1.18	49.07	2.99	0.16	**31.10**	1.95	0.28
	15	41.59	4.03	1.22	36.44	3.15	0.19	**21.83**	1.65	0.31
	20	32.55	3.86	1.26	26.17	2.36	0.22	**14.64**	1.26	0.31
	40	14.43	2.11	1.36	12.51	0.83	0.33	**11.47**	0.20	0.31
500	15	106.87	5.03	12.69	94.38	5.56	1.63	**44.60**	3.19	2.92
	30	58.52	7.10	14.65	46.51	4.13	2.80	**25.32**	2.03	3.25
	45	32.23	5.66	16.66	25.63	2.18	4.25	**17.82**	0.78	3.14
	60	20.33	3.46	17.64	17.72	0.92	5.41	**16.03**	0.29	3.24
1000	20	217.71	9.48	150.06	195.96	7.97	13.18	**51.96**	1.81	41.30
	40	124.21	17.33	167.91	99.57	7.86	22.61	**33.52**	2.95	51.69
	60	69.83	12.20	183.21	50.97	5.24	33.28	**27.75**	1.72	54.70
	100	28.95	4.14	189.33	23.41	0.78	55.34	**22.58**	0.19	55.63

The mean BDST tree costs obtained over a wide range of Euclidean instances for the heuristics clearly show that the CBLSoC heuristic outperforms the other two heuristics on all instances. The results obtained also indicate lower standard deviation as compared to the other two heuristics.

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

Comparing the mean BDST costs obtained in table 1 and those given for the RTC heuristic in table 2, it can be seen that on small diameter bounds, the RTC heuristic performs significantly better than the deterministic heuristics discussed above, and presented in table 1. However, as the diameter bound is increased, RTC by virtue of the fact that its edge selection is always random, fails to take advantage of the constraint relaxation and, as a result, is able to produce only slight improvements in tree costs. By contrast, the OTTC, CBTC and CBLSoC produce lower cost trees with larger diameter bounds; the proposed CBLSoC heuristic produces better trees than the other two on every instance size considered.

Table 2. Results for the RTC and NQC Heuristics on benchmarks graphs up to 1000 nodes

Instances		RTC			NQC-LSoC			NQC-Greedy		
n	D	X	SD	t	X	SD	t	X	SD	t
100	5	15.30	0.78	<0.01	**14.64**	0.53	0.0003	**14.64**	0.55	0.0033
	10	**9.38**	1.80	0.01	10.10	0.32	0.0003	9.95	0.32	0.0060
	15	**9.25**	0.28	0.01	9.65	0.32	0.0003	9.26	0.33	0.0060
	25	**9.19**	0.27	0.01	9.47	0.28	0.0003	**9.03**	0.30	0.0060
250	10	**16.80**	0.33	0.16	**16.76**	0.44	0.0020	17.01	0.54	0.0440
	15	15.30	0.23	0.25	**15.10**	0.35	0.0020	**14.70**	0.34	0.0547
	20	15.05	0.23	0.27	**14.36**	0.37	0.0063	**13.49**	0.23	0.0623
	40	14.98	0.21	0.27	**14.07**	0.19	0.0020	**13.34**	0.17	0.0530
500	15	**22.26**	0.37	4.47	23.01	0.59	0.0253	24.19	0.76	0.2763
	30	21.54	0.33	5.30	**19.87**	0.42	0.0090	**18.52**	0.35	0.3440
	45	21.43	0.33	5.43	**19.21**	0.55	0.0120	**17.83**	0.32	0.3233
	60	21.45	0.34	5.29	**18.83**	0.34	0.0270	**17.69**	0.29	0.3347
1000	20	**31.15**	0.24	52.28	32.04	0.88	0.0580	36.10	1.66	3.1853
	40	30.85	0.22	56.17	**27.50**	0.68	0.0883	**25.29**	0.48	3.9543
	60	30.84	0.24	56.82	**26.68**	0.66	0.0863	**24.33**	0.23	4.0297
	100	30.84	0.24	55.51	**25.49**	0.20	0.0780	**23.87**	0.18	4.0510

Table 2 also provides the results of the greedy and LSoC-based NQC Heuristic variants alongside those given for the RTC heuristic. As can be seen from the documented mean tree costs, the NQC heuristics produce lesser cost trees on average in several instances. The greedy variant obtains the best costs, in time that is $O(n^3)$, which is the same as the running time of the RTC heuristic. However, the LSoC-based variant also produces better trees in comparison to the RTC in several instances, especially on the higher sized instances and with increasing values of diameter constraints. Furthermore, the LSoC-based variant of the NQC heuristic runs in $O(n^2)$ time, which is an order of significance lesser than both the greedy variant and the extant RTC heuristic.

4 Conclusions

This work presents novel deterministic heuristics that produce good approximate BDMSTs in $O(n^3)$ and $O(n^2)$ time. These are compared with three existing heuristics for the problem over a large set of Euclidean benchmark instances. The Center-based

LSoC heuristic performs better than the relatively more greedy OTTC heuristic, and runs in time which is O(n) lesser. This heuristic also produces consistently better results in comparison to the CBTC heuristic. In the case of instances with small diameter bounds, all three heuristics produce inferior quality trees in comparison with the randomized RTC heuristic; however, they obtain better mean costs vis-à-vis RTC as the diameter bound is raised. The quadrant centers-based heuristics presented were developed keeping in mind the drawbacks encountered with the OTTC heuristic. They are aimed specifically at Euclidean problem instances; the nine quadrants-based approach was found to produce superior results as compared to other empirical approaches explored. The greedy variant of the NQC heuristic produces the lowest mean costs on several of the instances, however it runs in $O(n^3)$ time. Significantly, the LSoC-based variant of the NQC heuristic performs competitively in $O(n^2)$ time, obtaining mean costs comparable with the RTC and NQC-Greedy heuristics on small diameter bounds, and lower cost BDSTs in comparison with the OTTC, CBTC and CBLSoC heuristics on several instances with larger diameter bounds.

Acknowledgments. The authors are extremely grateful to the Revered Chairman, Advisory Committee on Education, Dayalbagh for the continued guidance. The authors also thank Prof. Bryant A. Julstrom, St. Cloud University, Minnesota (USA) for providing the extended suite of problem instances for the BDMST problem.

References

1. Abdalla, A., Deo, N., Gupta, P.: Random-tree Diameter and the diameter constrained MST. In: Congressus Numerantium, vol. 144, pp. 161–182. Utilitas Mathematica (2000)
2. Bala, K., Petropoulos, K., Stern, T.E.: Multicasting in a linear lightwave network. In: IEEE INFOCOM 1993, pp. 1350–1358 (1993)
3. Bookstein, A., Klein, S.T.: Compression of correlated bit-vectors. Information Systems 16(4), 110–118 (1996)
4. Raymond, K.: A tree-based algorithm for distributed mutual exclusion. ACM Transactions on Computer Systems 7(1), 61–77 (1989)
5. Garey, M.R., Johnson, D.S.: Computers and Intractibility: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
6. Kortsarz, G., Peleg, D.: Approximating shallow-light trees. In: Proc. Eighth ACM-SIAM Symposium on Discrete Algorithms, pp. 103–110 (1997)
7. Julstrom, B.A., Raidl, G.R.: A permutation-coded EA for the BDMST problem. In: GECCO 2003 Workshops Proc., Workshop on Analysis & Design of Representations, pp. 2–7 (2003)
8. Prim, R.C.: Shortest connection networks and some generalizations. Bell System Technical Journal 36, 1389–1401 (1957)
9. Julstrom, B.A.: Greedy Heuristics for the Bounded Diameter Minimum Spanning Tree Problem. ACM J. Exp. Algor. 14, Article 1.1 (2009)
10. Achuthan, N.R., Caccetta, L., Cacetta, P., Geelen, J.F.: Algorithms for the minimum weight spanning tree with bounded diameter problem. Optimization: Techniques and Applications, 297–304 (1992)
11. Singh, A., Gupta, A.K.: Improved heuristics for the bounded diameter minimum spanning tree problem. Soft Computing 11(10), 911–921 (2007)