Dynamic Capacitated Minimum Spanning Trees

Raja Jothi and Balaji Raghavachari
Department of Computer Science, University of Texas at Dallas
Richardson, TX 75083, USA
{raja, rbk}@utdallas.edu

***Abstract*— Given a set of terminals, each associated with a positive number denoting the traffic to be routed to a central terminal (root), the *Capacitated Minimum Spanning Tree* (CMST) problem asks for a minimum spanning tree, spanning all terminals, such that the amount of traffic routed from a subtree, linked to the root by an edge, does not exceed the given capacity constraint k. The CMST problem is NP-complete and has been extensively studied for the past 40 years. Current best heuristics, in terms of cost and computation time ($O(n^2 \log n)$), are due to Esau and Williams [1], and Jothi and Raghavachari [2].**

In this paper, we consider the *Dynamic Capacitated Minimum Spanning Tree* (DCMST) problem in which a CMST T, spanning n nodes and rooted at a central node r, with capacity constraint k is given as part of the input. Requests, from new nodes, for joining T arrive dynamically (online), one at a time. Under this setting, we want to add the new node to the existing CMST without having to recompute the entire CMST, which would take $O(n^2 \log n)$ time. To our knowledge, the DCMST problem has not been studied before. For the DCMST problem, we propose three methods (COD, CFOD and BSA) to add the new nodes to the existing CMST. All our heuristics run in linear time. We compare the performance of our heuristics with a CMST algorithm that recomputes the solution every time a new node arrives.

***Keywords*—Capacitated Minimum Spanning Trees, Network Design, Dynamic Graph Algorithms.**

I. INTRODUCTION

Many network design problems involve finding minimum cost spanning trees satisfying certain connectivity constraints. Unfortunately, many such problems are intractable (NP-hard). Due to this reason, the research community has turned its attention on finding fast and efficient heuristics to solve these computationally “hard” problems.

One of the well-studied problems in the field of telecommunications is the *Capacitated Minimum Spanning Tree* (CMST) problem. Given a set of terminals, each associated with a positive number denoting the traffic to be routed to the central terminal (root), the CMST problem asks for a minimum spanning tree, spanning all terminals, such that the amount of traffic routed from a subtree, linked to the root by an edge, does not exceed the given capacity constraint k. Each edge connecting two nodes have a cost associated with it. Usually, the cost of a edge is proportional to its length. The CMST problem is NP-hard [3], [4]. In the design of telecommunication networks, the CMST problem corresponds to the designing of a minimum cost tree network by installing expensive (fiber-optic) cables along its edges, with a capacity constraint k on the cable being used. The cables are assumed to be bought at unit cost per unit length. Since the cable capacity is k, every subtree, connected to the root using an edge, can route a traffic of at most k to the root. Throughout this paper, the terms terminals, nodes and vertices are used interchangeably. The CMST problem can formally be defined as follows:

CMST: Given an undirected vertex-weighted graph $G = (V, E)$, where V is the set of n nodes and E is the set of edges connecting the nodes, root node $r \in V$, and an integer $k > 0$. The CMST problem asks for a minimum spanning tree with the sum of the vertex weights of every subtree, connected to r by an edge, at most k.

The CMST problem has been extensively studied in Computer Science and Operations Research for the past 40 years. Several heuristics and exact algorithms have been proposed (see Section I-B for more details). The most popular and efficient heuristic for the CMST problem is due to Esau and Williams [1]. In recent work, Jothi and Raghavachari [2] proposed a new heuristic that performs better than that of Esau and Williams. Both heuristics run in $O(n^2 \log n)$ time. There are other heuristics [5], [6], [7] based on Tabu Search and Simulated Annealing, which perform better in terms of the quality of the solution, but their running times are much higher. For other heuristics and exact algorithms on the CMST problem, we refer the reader to [5]. For worst-case performance ratios and approximation algorithms on the CMST problem, we refer the reader to [8], [9], [10].

A. The DCMST Problem

Many problems in the field of communication networks are designed as graph problems. Even though, in most cases, the input graphs are static (remain unchanged), there are situations in which graphs are subject to discrete changes, such as addition and deletion of nodes or edges. Networks, represented as graphs, that experience changes in the topology are called dynamic networks. An interesting problem in dealing with dynamically changing networks is to perform the updates swiftly and efficiently without having to shutdown the entire system.

In real life situations, it is quite normal for new nodes to join the already available (or functioning) network. While it could be cost efficient to recompute the overall network topology due to the addition of the new node, it is more time consuming to do so. Also, it is desirable that joining of the new nodes causes less inconvenience to the already functioning connected nodes.

In this context, we consider the *Dynamic Capacitated Minimum Spanning Tree* (DCMST) problem. In the DCMST problem, a CMST T, spanning n nodes and rooted at a central node r, with capacity constraint k is given as part of the input. Requests, from new nodes which are not in T, wishing to join the existing CMST arrive dynamically (online), one at a time. Under this setting, we want to add the new nodes, one at a time as they arrive, to the existing CMST without having to recompute the entire CMST, which would take $O(n^2 \log n)$ time. Since every node in the CMST problem is associated a weight, which denotes the amount of traffic that is to be routed to the central node r, every new request comes with a weight value associated the node that is to be connected to T. To our knowledge, the DCMST problem has not been studied before. In this paper, we propose three methods to add the new node to the existing CMST. All our heuristics run in $O(n)$ time. We compare the results of our heuristics with the results obtained from recomputing the solution every time a new node expresses its interest to join the existing CMST.

B. Related Work

Algorithms for finding a minimum spanning tree (MST) are well known. Amato et al. [11] and Cattaneo et al. [12] studied dynamic minimum spanning tree algorithms for maintaining minimum spanning trees in dynamic graphs. Demetrescu and Italiano [13] presented a fully dynamic algorithm for maintaining all pair shortest paths in dynamic directed graphs with real-valued edge weights. Galil and Italiano [14], [15] presented fully dynamic algorithms for edge connectivity problems. For more details on dynamic graph algorithms, we refer the reader to [11], [16].

The remainder of this paper is organized as follows. In Section II, we present our heuristics for the DCMST problem. Section III contains the experimental results of the heuristics. Finally, Section IV contains our concluding remarks and directions for future research.

II. Heuristics for the DCMST problem

In this section, we present three heuristics for the DCMST problem. Any heuristic for the DCMST problem will have a worst-case running time of at least $\Omega(n)$. This is due to the reason that for every new node that wishes to join the existing CMST, it takes $\Omega(n)$ time to compute the distances between that node and the rest of the nodes in the existing CMST. If the distance vector is given as a part of the input along with the new node, then one can choose to connect the new node to an arbitrary node in the tree (provided the capacity constraint is not violated) or to the central node. This naive approach will make the running time constant, but the cost of the tree will increase uncontrollably. We consider the case when the distance vectors are not given as part of the input. All our heuristics run in linear time.

Before we proceed to the heuristics, we define a few terms. Let T be the CMST to which a new node is to be connected. Let r be the root node of T. Let k denote the capacity constraint on T. Let $d(i, j)$ denote the distance between nodes i and j. We use the term *cluster* to refer to the subtrees rooted at the children of r.

A. The closest-or-direct (COD) heuristic

The new node m is connected to the closest node v in the CMST T, provided that the sum of the vertex weights in the cluster containing v plus the vertex weight of m is at most k, and $d(m, v) < d(m, r)$. Otherwise, m is directly connected to r. The time to compute the distances between m and the nodes in T is linear. During the computation of distances between m and the nodes in T, m's closest node can be found. Once m's closest node is known, it takes constant time to connect m to T. Thus the overall time complexity of the COD heuristic is linear.

B. The closest feasible or direct (CFOD) heuristic

The new node m is connected to the closest node v in the CMST T, provided that the following two conditions are satisfied:

1) $d(m, v) < d(m, r)$.
2) the sum of the vertex weights in the cluster containing v plus the vertex weight of m is at most k.

If the first condition is not satisfied, m is directly connected to r. If the second condition is not satisfied, we repeat the above with m's second closest node and so on, until m is included in the tree.

The CFOD heuristic can be implemented in linear time. The time to compute the distances between m and the nodes in T is linear. During the computation of distances between m and the nodes in T, compute the value $MAX = \max\{d(m, r) - d(m, v)\}, \forall_{v \in T}$. If $MAX > 0$, then m is connected to node $v \in T$ that contributed to MAX, otherwise m is directly connected to r.

C. The best savings (BSA) heuristic

The new node m is connected to node v in the CMST T, which provides the greatest savings. This heuristic is in line with Esau and Williams algorithm [1]. Let C_p be the cluster containing p and $d(C_p, r)$ be the distance between r and its closest node in C_p. Let $s_{pq} = d(C_p, r) - d(p, q)$. During the computation of distances between m and the nodes in T, compute the savings s_{mv} and s_{vm} for every node $v \in T$. Keep track of the node $i \in T$ for which the savings produced (either s_{im} or s_{mi}) is positive and maximum under the condition that the sum of vertex weights in C_i plus the vertex weight of m is at most k. Connect i and m if the savings are positive, else connect m to r directly.

The time to compute the distances between m and the nodes in T is linear. Since the node in T to which m is connected is found during the computation of distances between m and the nodes in T, the running time of BSA heuristic is linear.

III. Experimental Results

We conducted simulations to study the performance of our heuristics against a CMST heuristic (Esau and Williams algorithm [1]) which recomputes the solution every time a new node has to be connected to the already existing CMST. Obviously recomputing the CMST will produce better solutions when compared to our heuristics. But since the goal behind dynamic graph algorithms is to perform updates without having to recompute the entire solution, running time is very crucial, apart from not disturbing the operation of the existing nodes considerably. Since there is no proper lower bound with which one can compare dynamic online algorithms, it is of normal practice to compare dynamic online algorithms with offline algorithms which recompute the entire solution.

Each input instance contains a CMST of 100 nodes with capacity constraint k, which was computed using the Esau and Williams algorithm [1]. The nodes in the CMST were generated using a random distribution of points in a 100×100 grid. The cost of an edge that connects points i and j is the Euclidean distance between i and j. For each instance, 50 new nodes were generated uniformly at random. Every time a new node is generated, it has to be connected to the CMST. We test our heuristics for both weighted and unweighted versions of the DCMST problem. In the unweighted DCMST version, the weights of all the node is set to 1. That is, each node has a traffic of 1 unit that has to be routed to the root node. In the weighted DCMST version, each node is associated with an arbitrary weight less than or equal to the capacity constraint k.

For each k value, we generated 100 instances, with each instance starting with a 100 node CMST and performing addition of 50 new nodes. For each k value, we computed the average cost of the tree after the addition of i^{th} new node, where $i = 1, \ldots, 50$. Table I shows the average time to perform the addition of new node to the existing tree. Figures 1 to 8 compare the changes in the cost of the tree for all the three heuristics proposed in this paper and the Esau and Williams (EW) algorithm [1]. Our experimental results show that CFOD and BSA produce the best results among the three proposed heuristics. In terms of time, CFOD is little faster than BSA. The fastest in terms of time is the COD heuristic, but it does not produce good results. While the worst-case running times of the three proposed heuristics are linear, the worst-case running time of EW algorithm is $(n^2 \log n)$.

IV. Conclusion

In this paper, we considered the dynamic capacitated minimum spanning tree problem. We presented three heuristics for the DCMST problem and compared their performance with the CMST algorithm (Esau and Williams) that recomputes the solution every time a new node arrives. Our experiments show that two of our heuristics (CFOD and BSA) produce the best results.The CFOD heurstic is slightly faster than the BSA heuristic. Our other heuristic (COD) is the fastest among the three heuristics, but the quality of the solutions were not as good as that of CFOD and BSA. The worst-case running times of all our heuristics are linear. We are currently investigating several other heuristics for the DCMST problem.

Acknowledgment

This research was supported by the National Science Foundation under grant CCR-9820902.

References

[1] L.R. Esau and K.C. Williams, "On teleprocessing system design," *IBM Sys. Journal*, Vol. 5, pp. 142-147, 1966.

[2] R. Jothi and B. Raghavachari, "Design of local access networks," to appear in *Proc. of the 15th IASTED Intl. Conf. on Parallel and Distributed Comput. and Systems* (PDCS), 2003.

[3] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W.H. Freeman, San Francisco, 1979.

[4] C.H. Papadimitriou, "The complexity of the capacitated tree problem," *Networks*, **8**, pp. 217-230, 1978.

[5] A. Amberg, W. Domschke and S. Voß, "Capacitated minimum spanning trees: Algorithms using intelligent search," *Comb. Opt.: Theory and Practice*, **1**, pp. 9-39, 1996.

[6] R.K. Ahuja, J.B. Orlin and D. Sharma, "A composite neighborhood search algorithm for the capacitated minimum spanning tree problem," Manuscript, 2001.

[7] Y.M. Sharaiha, M. Gendreau, G. Laporte and I.H. Osman, "A tabu search algorithm for the capacitated shortest spanning tree problem," *Networks*, **29**, pp. 161-171, 1997.

[8] K. Altinkemer and B. Gavish, "Heuristics with constant error guarantees for the design of tree networks," *Management Science*, Vol. 34, pp. 331-341, 1988.

[9] R. Jothi and B. Raghavachari, "Topological design of centralized communication networks," Manuscript, 2003.

[10] R. Jothi and B. Raghavachari, "Approximation algorithms for the capacitated minimum spanning tree problem and its variants in network design," Manuscript, 2003.

[11] G. Amato, G. Cattaneo and G.F. Italiano, "Experimental analysis of dynamic minimum spanning tree algorithms," *Proc. 8th ACM-SIAM Annual Symp. on Disc. Algorithms* (SODA), pp. 314–323, 1997.

[12] G. Cattaneo, P. Faruolo, U. Ferraro Petrillo and G.F. Italiano, "Maintaining dynamic minimum spanning trees: An experimental study," *Proc. 4th Workshop on Alg. Engg. and Experiments*, 2002.

[13] C. Demetrescu and G.F. Italiano, "Fully dynamic all pairs shortest paths with real edge weights," *Proc. 42nd IEEE Annual Symp. on Foundations of Computer Science* (FOCS), pp. 260-267, 2001.

[14] Z. Galil and G.F. Italiano, "Fully dynamic algorithms for edge connectivity problems," *ACM Symposium on Theory of Computing*, pp. 317-327, 1991.

[15] Z. Galil and G.F. Italiano, "Fully dynamic algorithms for 2-edge-connectivity," *SIAM J. Computing*, Vol. 21, pp. 1047-1069, 1992.

[16] D. Eppstein, Z. Galil and G.F. Italiano, "Dynamic graph algorithms," *Algorithms and Theory of Computation Handbook*, CRC Press, 1999.

[17] D. Alberts, G. Cattaneo and G.F. Italiano, "An experimental study of dynamic graph algorithms," *ACM Journal on Experimental Algorithmics*, Vol. 2, 1997.

		Time (in milliseconds)			
Capacity constraint k	Unit weight vertices?	EW	COD	CFOD	BSA
3	Yes	235.49	0.04	0.18	0.46
5	Yes	149.88	0.00	0.10	0.30
10	Yes	95.33	0.04	0.12	0.28
20	Yes	64.40	0.06	0.06	0.22
100	No	206.59	0.02	0.14	0.44
200	No	192.41	0.00	0.14	0.36
300	No	158.02	0.08	0.14	0.34
400	No	134.86	0.02	0.04	0.24

TABLE I
AVERAGE RUNNING TIME FOR EACH UPDATE.

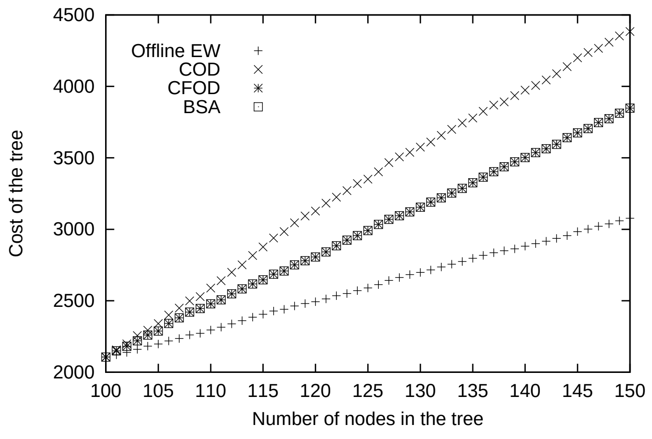

Fig. 1. Average cost of the tree after each node update (unit weight nodes with $k = 3$).

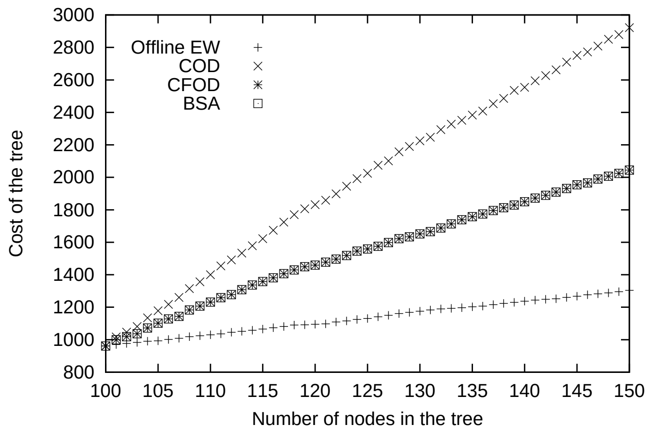

Fig. 3. Average cost of the tree after each node update (unit weight nodes with $k = 10$.)

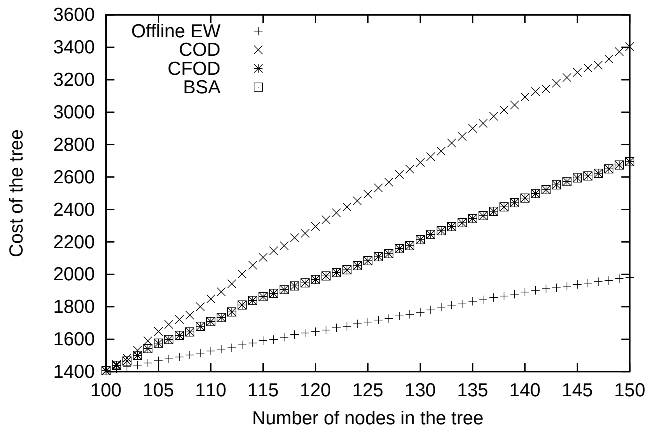

Fig. 2. Average cost of the tree after each node update (unit weight nodes with $k = 5$.)

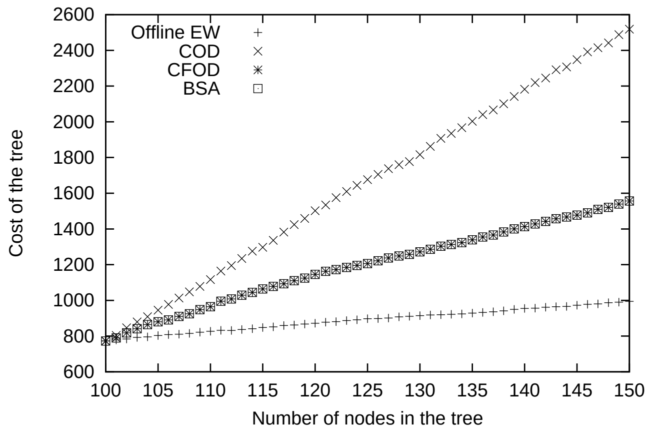

Fig. 4. Average cost of the tree after each node update (unit weight nodes with $k = 20$.)

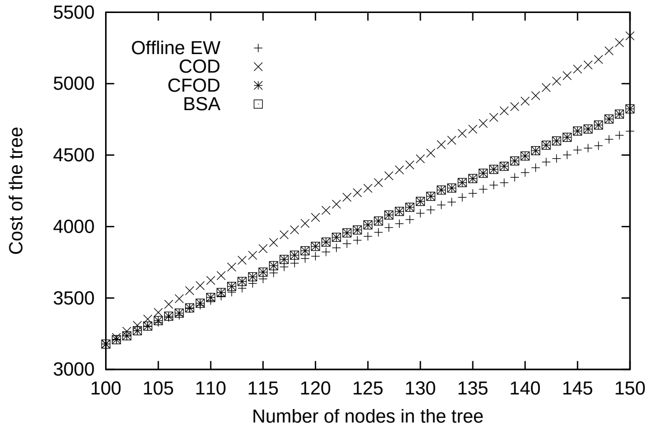

Fig. 5. Average cost of the tree after each node update (arbitrary weighted nodes with $k = 100$.)

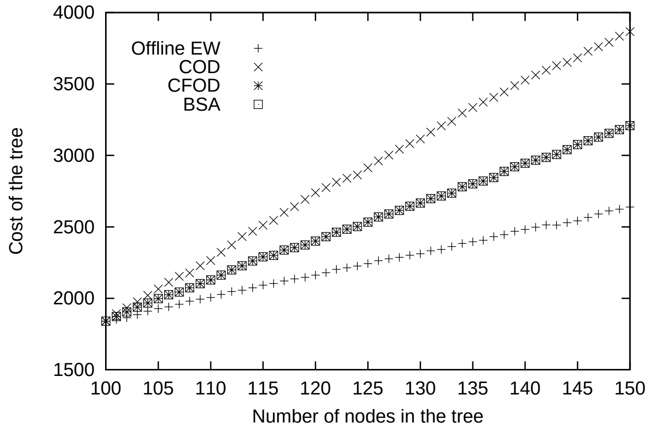

Fig. 6. Average cost of the tree after each node update (arbitrary weighted nodes with $k = 200$.)

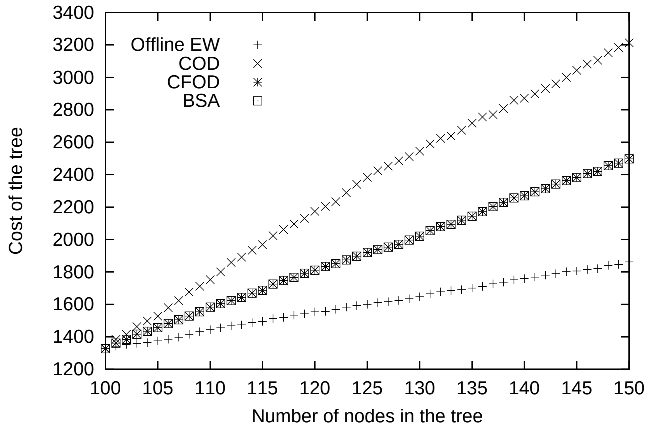

Fig. 7. Average cost of the tree after each node update (arbitrary weighted nodes with $k = 300$.)

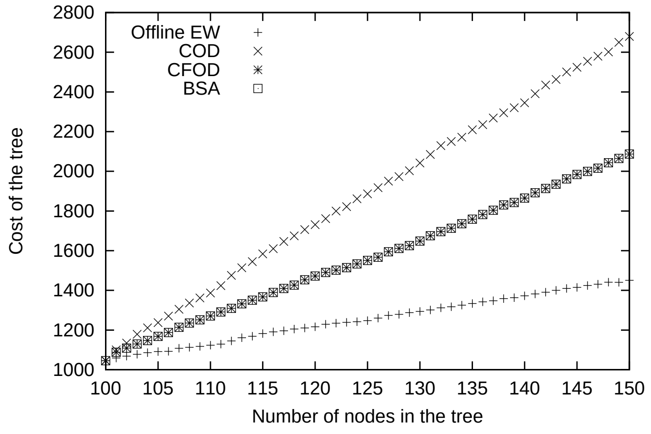

Fig. 8. Average cost of the tree after each node update (arbitrary weighted nodes with $k = 400$.)