THE CAPACITATED MINIMUM SPANNING TREE

by

K. M. Chandy and Tachen Lo

TR-17

May 1973

THE CAPACITATED MINIMUM SPANNING TREE *

by
K. M. Chandy
and
Tachen Lo
The Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

ABSTRACT

The capacitated minimum spanning tree is an offspring of the minimum spanning tree and network flow problems. It has application in the design of multipoint linkages in elementary teleprocessing tree networks. Some theorems are used in conjunction with Little's branch and bound algorithm to obtain optimal solutions. Computational results are provided to show that the problem is tractable.

This research was supported by NSF Grant GJ - 35109

INTRODUCTION

The minimum spanning tree problem is well known [2]. Given a non-directional graph with costs associated with the edges, the problem is to construct a minimum cost spanning tree. The capacitated minimum spanning tree is a related problem which is also related to some network flow problems. One of the vertices in the graph is a sink, and all other vertices are sources. Associated with each source i is a surplus a_i (of some commodity), and a demand (equal to the sum of the surpluses) is associated with the sink. The problem is to construct a spanning tree at minimum cost in which there is commodity flow from the sources to the sink only along edges of the tree, such that the surplus is shipped out of every source and such that the flow on each edge of the tree does not exceed a given capacity c. This problem has application in teleprocessing systems design [3,4,5,6].

Let d_{ij} be the cost of the edge between vertices i and j. We shall assume that the sink is always vertex 1. Consider the example shown in Table 1. The minimum spanning tree shown in Fig. 1(a) is not feasible in the capacitated problem since the flow from vertex 2 to 1 exceeds capacity. The optimal solution is shown in Fig. 1(b).

Insert Table 1 about here

Insert Fig. 1 about here

In teleprocessing applications, the sources are data terminals and the sink is the data processing center. The terminals communicate with the center along communication links. The "surplus" at a terminal is the amount of traffic it generates. The capacity of a link is the maximum traffic the link can carry while maintaining acceptable response times. We assume that there is a unique path from each terminal to the data processing center (i.e. the network must be a tree) since traffic from a single terminal cannot be simultaneously transmitted along several links without additional synchronizing circuitry.

Earlier Work

The algorithm used here is an application of the branch and bound algorithm developed by Little et al.[1]. We develop some theorems and modify the Little algorithm to aid in the search for the optimal solution. The problem can be cast as an integer programming problem if the surpluses are integers [3]. However, the number of variables and constraints is large, even for medium sized (50 vertex) problems.

Karnaugh [4], Martin [5] and Esau-Williams [6] have developed excellent heuristics for the problem. Chandy and Russell [3] developed an optimal algorithm and showed that the Martin and Esau-Williams heuristics were near-optimal. Karnaugh's heuristic is also near-optimal since it is more accurate (though it does require more computation time) than Esau-Williams. The algorithm presented here is an order of magnitude faster than the one in [3].

The optimal algorithm presented here is not intended to compete with the heuristics. From a pragmatic point of view it is better to use the heuristics than a slower optimal algorithm since approximations in the data make small deviations from optimality unimportant. Our

algorithm is presented to show that the capacitated minimum spanning tree is indeed tractable.

THEORY

For ease of exposition we assume that the minimum spanning tree and capacitated minimum spanning tree are unique. However our algorithm is perfectly general.

We shall denote the edge between vertices i and j as (i,j). Let V be the set of vertices in an undirected graph. Let T be the minimum spanning tree on V. Let T include edge (i,j). Let W be any subset of V which includes both vertices i and j. We define the minimum spanning tree over W as the minimum cost spanning tree which spans all vertices in W and does not span any vertex not in W.

Lemma 1: The minimum spanning tree over W includes edge (i,j).

Proof: Follows from the "greedy" algorithm [7].

Let T be any tree over V. Let T include edge (i,1). We define a major branch of T on edge (i,1) as that subgraph of T which consists of all vertices which are connected to vertex 1 through edge (i,1). In Fig. 1(b), the major branch on edge (2,1) consists of edges (2,1) and (4,2), while the major branch on edge (3,1) consists of edges (3,1) and (5,3). We shall define a vertex cluster of a major branch as the set of sources spanned by the major branch. The vertex cluster of the major branch on edge (2,1) consists of vertices 2 and 4.

Lemma 2: Every major branch of a capacitated minimum spanning tree is a minimum spanning tree.

In Fig. 1(b) the major branch consisting of edges (2,1), (4,2) is the minimum spanning tree over vertices 1,2,4.

Corollary 1 If the minimum spanning tree over the set of all vertices V includes edge (i,j) and if vertices i and j belong to the same vertex cluster of the capacitated minimum spanning tree then the capacitated minimum spanning tree includes edge (i,j).

Corollary 2 If the minimum spanning tree over V includes edge (i,1) then so does the capacitated minimum spanning tree.

THE ALGORITHM

The algorithm is a modified version of Little's procedure [1]. We assume familiarity with [1]. The results of the previous section are used to speed up the search for the optimal solution.

The cost of the minimum spanning tree is used as a lower bound on the cost of the capacitated minimum spanning tree. Branching is similar to Little's algorithm: we partition a set of solutions A into subsets B and B' where two vertices i and j are required to be in the same cluster in subset B and not allowed to be in the same cluster in B'. From corollary 1 if the minimum spanning tree includes edge (i,j) then the capacitated minimum spanning tree in subset B includes (i,j). A lower bound for B' is obtained by setting the costs of all edges between the cluster that includes vertex i and the cluster that includes vertex j to infinity.

The algorithm will be illustrated using the example shown in Table 1. The branch and bound tree for this example is shown in Fig. 2 and Table 2. We first construct a minimum spanning tree to obtain a lower bound on the cost of the capacitated minimum spanning tree. The minimum spanning tree and the lower bound are associated with the root node. From corollary 2, since the minimum spanning tree includes edge (2,1) we may restrict our search to trees which

include (2,1).

We branch from the root node into two nodes A and A' where vertices 4 and 2 are required to be in the same cluster in node A. From corollary 1, we may restrict attention in node A to trees which include edges (4,2) and(2,1). If vertices 2 and 4 belong to the same cluster, the flow along the edge connecting the cluster to the sink will be at least $a_2 + a_4 = 4$. Hence connecting vertex 5 to this cluster will cause the flow in that edge to be at least $a_2 + a_4 + a_5$ which exceeds the given capacity. Hence in node A and all of its descendent nodes we do not allow edges between the cluster that includes vertex 5 and the cluster that includes 2 and 4. In other words, for node A, we set the costs of edges (5,2) and (5,4) to infinity.

The only difference between this algorithm and a direct application of the Little algorithm as described in [3] is the use of clusters. However, it is the notion of clusters that makes the problem tractable. Since the reader (familiar with Little's algorithm) will have no trouble following our algorithm we shall stop discussing the example: Fig. 2 should be clear.

COMPUTATIONAL RESULTS

The optimal branch and bound algorithm was compared with a heuristic suggested by Martin [6,3]. The algorithms were coded in Fortran and run on a CDC 6600 at the University of Texas at Austin. The CPU time in seconds required to execute the algorithms was measured. Twelve problems each of size 10, 20, 30 and 40 vertices (a total of forty eight) were generated at random and solved. The data for a problem consists of a graph showing the vertices (sources and the link) distances, and surpluses at each node and the link capacity. The vertices were placed at random

on a two dimensional Cartesian plane; the coordinates for each vertex and the surplus at each source were generated by an independent uniform random number generator. For each problem size (in terms of number of vertices) three graphs were generated; for each graph the link capacity was set to K times the total surplus at all sources, where K was 2/3, 1/2, 1/3 and 1/6.

Tables 3 and 4 and Figure 3 present computational results. The execution time for the heuristic is much smaller than the optimal algorithm and is much less sensitive to problem structure, i.e., number of vertices, link capacity, distances between vertices etc. In 77% of the cases, the networks designed by the heuristic were not optimal. However, the costs of the network designed by the heuristic were very close to the optimal solution. Though in one case the cost obtained by the heuristic was 29% more than the optimal, the average deviation was under 5%.

CONCLUSION

An optimal algorithm for the capacitated minimum spanning tree problem was presented. Computational results show that the problem is indeed tractable. However, the CPU time required by the optimal algorithm is very sensitive to problem structure. Comparisons with a very fast heuristic developed by Martin suggest that in many cases, pragmatism dictates the use of a heuristic in preference to an optimal algorithm.

ACKNOWLEDGEMENT

The authors wish to thank Dr. Maurice Karnaugh who showed us that the execution time for the optimal algorithm was very sensitive to link capacity which led to experiments with different problem structures.

REFERENCES

1. J. D. C. Little, K. C. Murty, D. W. Sweeney, and C. Karel, "An Algorithm for the Traveling Salesman Problem," Operations Research, Vol. 11, pp 972-989, (November 1963).

2. E. W. Dijkstra, "A Note on Two Problems in Connection with Graphs," Numerische Mathematik, Vol. 1, pp. 269-271 (1959).

3. K. M. Chandy and R. A. Russell, "The Design of Multipoint Linkages in a Teleprocessing Tree Network," To appear IEEE Trans. Comp., Vol. C-21, No. 10, (October 1972).

4. Maurice Karnaugh, "Multipoint Network Layout Program," Report No. RC3723, Available from the author, IBM Thomas J. Watson Research Center, P. O. Box 218, Yorktown Heights, New York 10598.

5. L. R. Esau and K. C. Williams, "On Teleprocessing System Design," Part II, IBM Syst. Jour., Vol. 5, No. 3, pp 142-147 (1966).

6. J. Martin, Design of Real-Time Computer Systems, Prentice-Hall, Englewood Cliffs, N. J. (1969).

7. D. Gale, "Optimal Assignments in an Ordered Set: An Application of Matroid Theory," Journal Combinatorial Theory, Vol. 4, pp. 176-180 (1968).

D =

	1	2	3	4	5
1	-	1	3	3	4
2	1	-	1	1	3
3	3	1	-	2	1
4	3	1	2	-	3
5	4	3	1	3	-

Sink $a_1 = -8$ (demand: negative sign)

Sources $a_2 = 2$, $a_3 = 1$, $a_4 = 2$, $a_5 = 3$.

(surplus: positive sign)

Capacity = 5

TABLE 1 (Example)

DESCRIPTION OF THE BRANCH AND BOUND TREE

NODE	BOUND	EDGES INCLUDED	EDGES EXCLUDED	MIN. SPANNING TREE FOR GIVEN NODE	IS THIS TREE FEASIBLE?
ROOT	4	(2,1)	-	(2,1)(3,2)(4,2)(5,3)	NO
A	4	(4,2)	(5,4)(5,2)	(2,1)(3,2)(4,2)(5,3)	NO
A'	5	-	(4,2)	(2,1)(3,2)(4,3)(5,3)	NO
AB	7	(3,2)	(5,3)	(2,1)(3,2)(4,2)(5,1)	YES
AB'	6	-	(3,2)(3,4)	(2,1)(3,1)(4,2)(5,3)	YES
A'C	8	(3,2)	(4,3)(5,2)(5,3)	(2,1)(3,2)(4,1)(5,4)	YES
A'C'	7	-	(3,2)	(2,1)(3,1)(4,3)(5,3)	NO

Optimal Solution Corresponds to Node AB'

TABLE 2

CAP \ size / method \ time		10 nodes	20 nodes	30 nodes	40 nodes
$\frac{2}{3}\Sigma t_i$	Martin	0.016	0.04	0.082	0.116
	CMST	0.0784	2.327	2.435	2.523
$\frac{1}{2}\Sigma t_i$	Martin	0.015	0.024	0.050	0.090
	CMST	0.149	5.88	6.057	15.27
$\frac{1}{3}\Sigma t_i$	Martin	0.0*	0.032	0.051	0.090
	CMST	0.435	49.60	52.34	58.605
$\frac{1}{6}\Sigma t_i$	Martin	0.0*	0.032	0.047	0.085
	CMST	0.237	70.874	76.14	97.82**

Table 3 Average execution times for 48 problems

* Execution time is less than one millisecond

** In one case the execution time exceeds 100 seconds

node size \ CAP / dev.% in cost	$\frac{2}{3}\Sigma t_i$	$\frac{1}{2}\Sigma t_i$	$\frac{1}{3}\Sigma t_i$	$\frac{1}{6}\Sigma t_i$
10-node	9.9	7.69	6.33	4.45
20-node	2.33	2.78	4.91	5.05
30-node	3.26	1.43	4.57	1.23
40-node	1.96	3.61	3.09	2.44

Table 4 Average deviation in cost of the heuristic

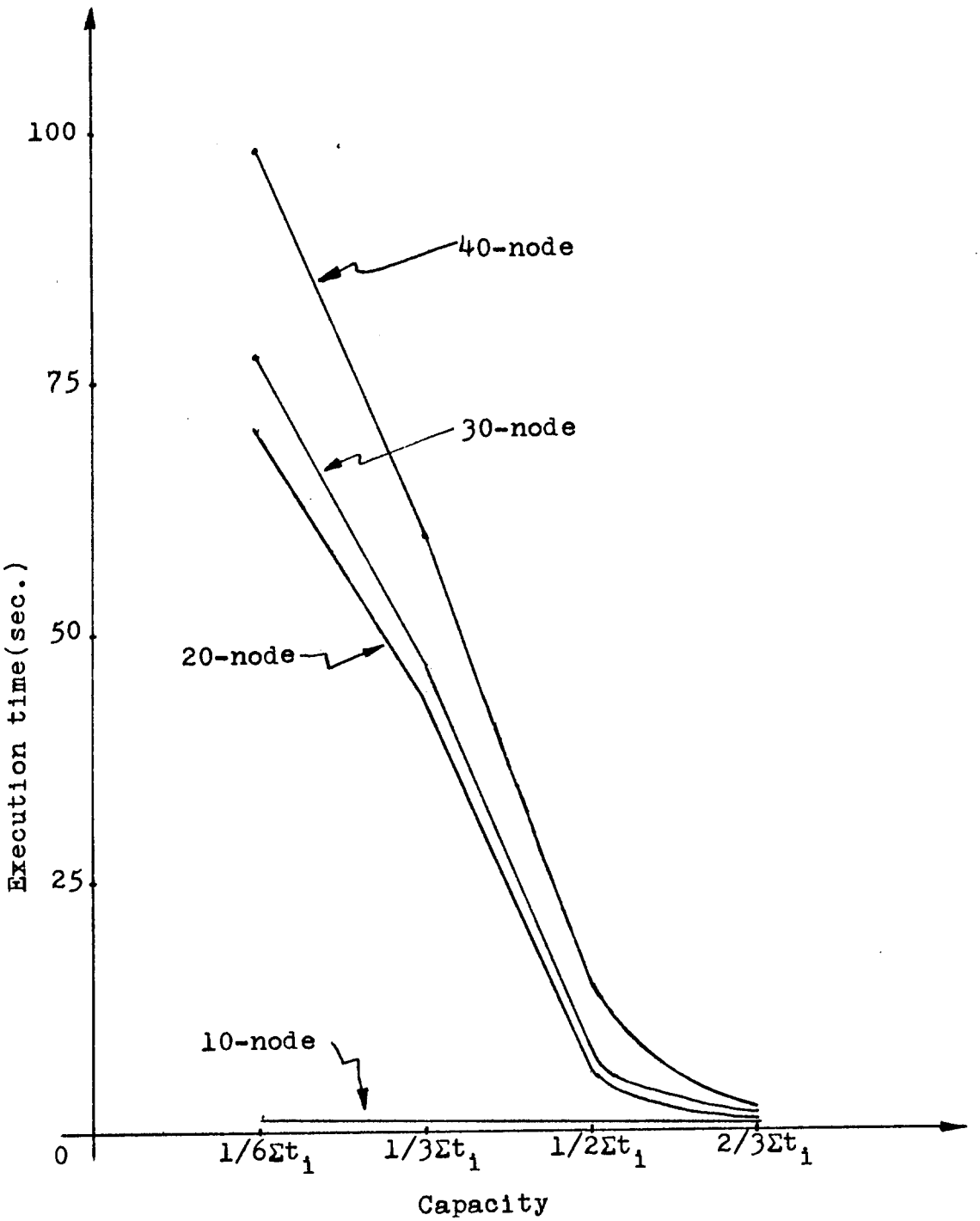

Figure 3 Graphical results for the execution time of the optimal algorithm