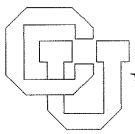
A Good Algorithm for Smallest Spanning Trees With a Degree Constraint

Harold N. Gabow

CU-CS-105-77 June 1977



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

A GOOD ALGORITHM FOR SMALLEST SPANNING TREES WITH A DEGREE CONSTRAINT

by

Harold N. Gabow
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309

June, 1977

Abstract

Given a connected graph with edge costs, we seek a spanning tree having a specified degree at one vertex r, with cost as small as possible. A previous algorithm, using edge exchanges, has run time $O(V^2)$; we improve this to $O(E \log\log V + V \log V)$. Here V and E are the number of vertices and edges. The algorithm uses edge exchanges ordered efficiently on a reduced graph; it also uses efficient algorithms for minimum spanning trees and priority queues.

1. Introduction

Many variations of the minimum spanning tree problem occur in communication networks and computer networks ([H,P]). These problems are often intractable ([G,P]). In contrast, the problem we discuss enjoys an efficient solution.

We are given a connected undirected graph G, a vertex r, and a positive integer b. Each edge e in G has a real-valued cost c(e). If T is a spanning tree, then d(r) denotes the degree of r in T; the cost of T is $c(T) = \sum_{e \in T} c(e)$. The problem is to find a spanning tree T with d(r) = b, having smallest cost among all such trees.

In an application, r might represent a central computing site; the other vertices are terminals which must be linked to r by cable paths. The degree constraint guarantees the computer's load is spread over a given number of ports. The smallest cost property guarantees as little cable as possible is used.

This problem was first discussed by Glover and Klingman [GK]. They sketch an algorithm that can be implemented in $O(V^2)$ time. We improve this to O(E log log V+V log V). Here V and E are the number of vertices and edges. Our algorithm is based on Glover and Klingman's technique of "admissible edge exchanges", coupled with a method for pruning the graph and efficiently ordering the exchanges. It also uses efficient algorithms for minimum spanning trees [CT,Y] and priority queues [AHU]. Section 2 derives some useful properties of spanning trees. Section 3 gives the algorithm and extends it to $\geq$ and $\leq$ degree constraints. Section 4 gives concluding remarks.

2. Optimal Edge Exchanges

The basic step of the algorithm is to make an edge exchange on a tree, i.e., starting with a spanning tree T, we replace an edge $e \in T$ by an edge $f \notin T$, obtaining a new tree T-e+f. This section describes how to choose the exchanges optimally. The results are implicit or actually stated in [GK]. Our development is self-contained and more direct.

Let R be the set of all edges incident to r in G. Let $\mathcal{T}_k$ be the set of all spanning trees T with $d(r)=|T \cap R|=k$. We seek a tree in $\mathcal{T}_b$ with smallest cost possible. For example, Figure 1(a) shows a graph, where an edge label both identifies the edge and specifies its cost. Figure 1(b) indicates a smallest tree T_k in each $\mathcal{T}_k$.

Note in general, for some $u \geq 1$, $\mathcal{T}_k \neq \emptyset$ exactly when $u \leq k \leq |R|$. For if T is a spanning tree with d(r)=u, any k-u edges incident to r can be added (and other edges deleted) to get a tree with d(r)=k. Note further, it is possible that u>1, if G-r is not connected.

In Figure 1, T_k and T_{k+1} are related by an edge exchange, e.g., $T_2=T_1-4+2$. We now prove this is true in general.

Theorem 1: Let T be a smallest tree in $\mathcal{T}_k$. Then

(a) If $\mathcal{T}_{k-1} \neq \emptyset$, there are edges $e \in T \cap R$, $f \notin T \cup R$, such that T-e+f is a smallest tree in $\mathcal{T}_{k-1}$;

(b) if $\mathcal{T}_{k+1} \neq \emptyset$, there are edges $e \in T-R$, $f \in R-T$ such that T-e+f is a smallest tree in $\mathcal{T}_{k+1}$.

Proof: We prove (a) only, since (b) is analogous. Let S be a smallest tree in $\mathcal{T}_{k-1}$ containing as many edges of T as possible. We will find edges e, f so S=T-e+f.

Choose f to be an edge in S-R-T. Note f exists, since $|S-R|>|T-R|$. To choose e, note there is a path in T joining the ends of f. Let e be an edge in this path that joins the two connected components of S-f. Thus both T-e+f and S-f+e are trees.

Suppose $e \notin R$. Then $T-e+f \in \mathcal{T}_k$, so $c(e) \leq c(f)$, and $S-f+e \in \mathcal{T}_{k-1}$, so $c(f) \leq c(e)$. We conclude $c(e)=c(f)$. Thus S-f+e is a smallest tree in $\mathcal{T}_{k-1}$ containing more edges of T than S does. This contradicts the definition of S, and shows $e \in R$.

Thus $T-e+f \in \mathcal{T}_{k-1}$, so $c(T-e+f) \geq c(S)$, and $S-f+e \in \mathcal{T}_k$, so $c(S-f+e) \geq c(T)$. The two inequalitites imply $c(T-e+f)=c(S)$, whence T-e+f=S, as desired. □

It is easy to characterize the exchanges of Theorem 1. We do this for part (a); a similar result holds for part (b), but is not needed here.

Corollary 1: Let T be a smallest tree in $\mathcal{T}_k$. If edges $e \in T \cap R$, $f \notin T \cup R$ are chosen so T-e+f is a tree and c(f)-c(e) is smallest among such edges, then T-e+f is a smallest tree in $\mathcal{T}_{k-1}$. □

We can also find a reduced set of edges that contains smallest trees in all $\mathcal{T}_k$'s. For this, recall a spanning forest of a graph consists of a spanning tree for each connected component.

Corollary 2: Let U be a minimum cost spanning forest of G-r. Then $U \cup R$ contains a smallest tree in $\mathcal{T}_k$, if $\mathcal{T}_k \neq \emptyset$.

Proof: Any spanning tree of G contains an edge from r to each tree in U. So u, the number of trees in U, is the least k with $\mathcal{T}_k \neq \emptyset$. A smallest tree in $\mathcal{T}_u$ consists of U and the smallest edge joining r to each tree. Now proceed by induction on k, using Theorem 1(b). □

3. The Algorithm

The basic idea is to find a smallest spanning tree containing R; then execute exchanges as in Corollary 1, until d(r) decreases to b. To speed up the computation, we first find a minimum spanning forest U in G-r; then only edges in $U \cup R$ need be considered, as shown by Corollary 2.

To find the exchanges specified by Corollary 1, we use a system of priority queues. Let T be the spanning tree at a given point in the computation. Consider an edge $e \in T \cap R$. An exchange that removes e from T adds an edge f joining the two connected components of T-e. A priority queue F(e) stores all such edges f. The priority of f is its cost c(f). Thus the smallest edge f that can replace e is easily found. A priority queue X stores exchanges (e,f), where for each edge $e \in T \cap R$, f is the smallest edge that can replace e. The priority of (e,f) is c(f)-c(e). Thus the smallest exchange of Corollary 1 is easily found in X.

Now we state the algorithm in Pidgin ALGOL [AHU].

```
procedure D; begin comment Input to D consists of a graph G, a
    vertex r, and an integer b, where T_b ≠ ∅. Output is T, a smallest
    spanning tree with d(r)=b;
    comment reduce the graph;
1.  let U be a minimum spanning forest of graph G-r; let R be the set
    of edges of G incident to r;
2.  remove all edges from G except those in U∪R;
3.  let T be a smallest spanning tree containing R;
    comment initialize priority queues;
4.  make X an empty priority queue;
5.  for each edge e∈R do
```

```
        begin
6.      let F(e) be a priority queue containing all edges f∈U that join
        components of T-e;
7.      if F(e)≠∅ then begin let f be the smallest edge in F(e);
        add (e,f) to X end;
        end;
    comment reduce r's degree by exchanges;
8.  while d(r)>b do
        begin
9.      remove the smallest exchange (e,f) from X; remove f from F(e);
10.     let e' be the edge in R-e such that f joins the components of T-e';
11.     remove the exchange (e',f') containing e' from X;
        remove f from F(e');
12.     merge F(e) and F(e') into a new priority queue F(e');
13.     if F(e')≠∅ then begin let f' be the smallest edge in F(e');
        add (e',f') to X end;
14.     T←T-e+f;
        end;
    end;
```

For Figure 1, in lines 1-3, $U=\{3,4\}, R=T=\{1,2,6\}=T_3$. The first exchange, (6,3), changes T to T_2; then exchange (2,4) changes T to T_1. Figure 1 also shows a "greedy" approach, similar to Kruskal's spanning tree algorithm [K], does not work for our problem: Suppose we form T by choosing edges, smallest first, subject to the constraints that T is acyclic and $d(r)\leq b$. If $b\leq 2$, T_b is found. But for $b=3$, T_2 is found, so the method fails.

Theorem 2: Procedure D finds a smallest spanning tree with $d(r)=b$. The time is $O(E \log\log V + V \log V)$ and the space is $O(E+V)$.

Proof: Correctness of D follows from Section 2, if we show line 9 selects an exchange with smallest possible cost. To do this, consider any any edge $f \in U-T-R$. The path in T joining the ends of f contains two edges of R, since any cycle in $U \cup R$ contains two edges of R. So for exactly two edges $e \in T \cap R$, (e,f) is a valid exchange. By induction, $f \in F(e)$ for these two edges e and no others. So the F queues record all valid exchanges. It follows easily that X contains a smallest cost exchange, and line 9 works correctly.

Now we analyze the time. The forest U is found (step 1) in O(E log log V) time, by using an efficient minimum spanning tree algorithm [CT,Y]. Tree T is found (step 3) by starting with edges $U \cup R$, contracting R into a single vertex, finding a minimum spanning tree, and then restoring R. The time is O(V log log V). (This can be reduced to O(V) using the techniques of [PS].).

The priority queue operations are adding an element (steps 6, 7, 13), finding and removing a smallest element (steps 7, 9, 13), removing an arbitrary element (twice in step 11), and merging two queues (step 12). Using 2-3 trees [AHU], a sequence of n such operations can be done in O(n log n) time. The algorithm does $O(|R|)=O(V)$ such operations, so the time is O(V log V). Note to do the remove operations in step 11, we maintain pointers from each edge e to its exchange (e,f) in X, and from each edge f to its two occurrences in queues F(e). The pointers for f are also used to do step 10 (once) in O(log V) time.

The time for finding spanning trees and manipulating priority queues dominates, so the time bound follows. The space bound is obvious. □

The algorithm is easily modified to find a smallest spanning tree with $d(r) \geq b$ (or $d(r) \leq b$). Suppose the test in line 8 is changed to $X \neq \emptyset$. Then a smallest tree in every T_k is found. The desired tree is the smallest in any T_k, $k \geq b$ ($k \leq b$). This tree is easily remembered.

Corollary 3: A smallest spanning tree with $d(r) \geq b$ ($d(r) \leq b$) can be found in time $O(E \log \log V + V \log V)$, and space $O(E+V)$. □

This approach can be slightly improved by observing how the cost of a smallest tree in T_k varies with k. In Figure 1, the minimum occurs in T_2. This illustrates the situation in general.

Corollary 4: The cost of a smallest spanning tree in T_k is a concave function of k.

Proof: Suppose T is a smallest tree in T_k, and the algorithm finds the exchange (e,f) (in line 9) giving a smallest tree in T_{k-1}. Note e' (in line 10) has $c(e') \leq c(e)$ (otherwise, the exchange (e',f) is chosen instead of (e,f).) An exchange that is valid for T-e+f but not valid for T has the form (e',g), where (e,g) is valid for T (see line 12). So for any exchange valid for T-e+f, there is a smaller exchange valid for T. Thus the cost of the exchange found in line 9 never decreases as k decreases. □

Corollary 4 can be used to reduce the number of iterations of the loop in lines 8-14, when searching for a smallest tree with $d(r) \geq b$ ($d(r) \leq b$). However the asymptotic time bound does not change.

4. Concluding Remarks

We have presented an algorithm for finding a smallest spanning tree subject to a constraint of the form $d(r)=b$, $d(r)\geq b$, or $d(r)\leq b$. We pose some related questions:

1. Is there a faster algorithm? In particular, can a smallest spanning tree with degree constraint be found as fast as a minimum spanning tree?
2. A minimum spanning tree in a directed graph can be found in $O(E \log V)$ time [T]. Can a smallest spanning tree with degree constraint be found efficiently in a directed graph?

The techniques presented here generalize to handle degree constraints at $O(\log V)$ different vertices efficiently; however $\Omega(n^{\varepsilon})$ constraints, for any $\varepsilon > 0$, lead to a problem that is intractable (i.e. NP-complete). We hope to report on these results and others in a forthcoming paper.

References

[AHU] A. Aho, J. Hopcroft and J. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

[CT] D. Cheriton and R. E. Tarjan, "Finding minimum spanning trees," SIAM J. on Computing 5 (1976), pp. 724-741.

[G] H. N. Gabow, "Hu's optimum communication spanning tree problem is NP-compete," unpublished memo.

[GK] F. Glover and D. Klingman, "Finding minimum spanning trees with a fixed number of links at a node", Report No. 74-5, Research Report CS #169, Center for Cybernetic Studies, University of Texas at Austin, Austin, Texas, 1974.

[H] T. C. Hu, "Optimum communication spanning trees," SIAM J. on Computing 3 (1974), pp. 188-195.

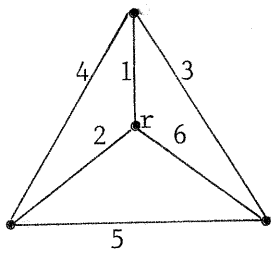
[K] J. B. Kruskal, Jr., "On the shortest spanning subtree of a graph and the traveling salesman problem," Proc. Amer. Math. Soc. 7 (1956), pp. 48-50.

[P] C. H. Papadimitriou, "The complexity of the capacitated tree problem," Tech. Rep. TR-21-76, Center for Research in Comp. Technology, Harvard U., Cambridge, Mass., 1976.

[SP] P. M. Spira and A. Pan, "On finding and updating spanning trees and shortest paths," SIAM J. on Computing 4 (1975), pp. 375-380.

[T] R. E. Tarjan, "Finding optimum branchings," Networks, to appear.

[Y] A. C. Yao, "An O(|E| log log |V|) algorithm for finding minimum spanning trees," Information Processing Letters 4 (1975), pp. 21-23.

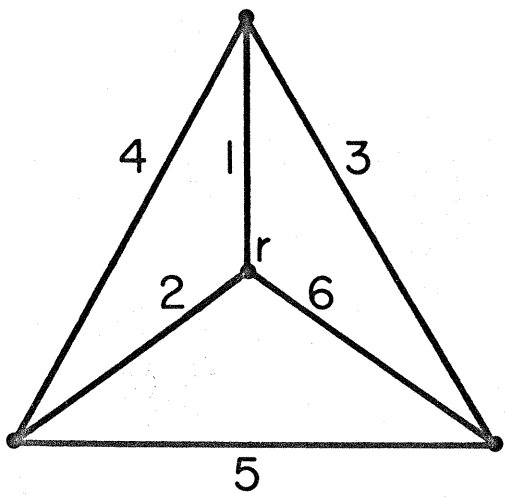


(a)

k	T_k	$c(T_k)$
1	1,3,4	8
2	1,2,3	6
3	1,2,6	9

(b)

Figure 1: Graph (a) and smallest tree T_k in T_k(b).


4
1
3
r
2
6
5