A LINEAR-TIME ALGORITHM FOR FINDING A MINIMUM SPANNING PSEUDOFOREST

Harold N. Gabow
Robert E. Tarjan

CS-TR-108-87

July 1987

A Linear-Time Algorithm for Finding a Minimum Spanning Pseudoforest

Harold N. Gabow[1]
Department of Computer Science
University of Colorado
Boulder, CO
80309

Robert E. Tarjan[2]
Computer Science Department
Princeton University
Princeton, NJ 08544
and
AT&T Bell Laboratories
Murray Hill, NJ 07974

July, 1987

Abstract.

A *pseudoforest* is a graph each of whose connected components is a tree or a tree plus an edge; a *spanning pseudoforest* of a graph contains the greatest number of edges possible. This paper shows that a minimum cost spanning pseudoforest of a graph with n vertices and m edges can be found in $O(m+n)$ time. This implies that a minimum spanning tree can be found in $O(m)$ time for graphs with girth at least $\log^{(i)} n$ for some constant i.

---

[1] Research supported in part by NSF Grant No. MCS-8302648 and AT&T Bell Laboratories.

[2] Research supported in part by NSF Grant No. DCR-8605962 and ONR Contract No. N00014-87-K-0467.

1. Introduction.

A *pseudotree* is a connected graph with equal number of vertices and edges, i.e., a tree plus an edge creating a cycle. A *pseudoforest* is a graph each of whose connected components has at least as many vertices as edges, i.e., each component is a tree or a pseudotree. Pseudoforests arise in many applications although the terminology is not standard. We use the terminology of [PQ], which uses pseudoforests to compute the density and arboricity of a graph; see [W] for refinements of this approach. Pseudotrees are essentially the *1-trees* used in [HK] to solve the traveling salesman problem. The directed version of a pseudoforest is called a *functional graph* in [Be], since it corresponds to the graph of a finite function. For this reason pseudoforests commonly arise in parallel processing, when each processor chooses a successor (e.g., [GPS]). The pseudoforests of a graph form the *bicircular matroid*, which is important in the study of rigidity of bar-and-body frameworks [WW]. In the problem of minimum cost network flow with losses and gains [L], a linear programming basis is a pseudoforest [D]. A pseudotree is also called a *unicyclic graph* [e.g., MH].

With these applications as motivation we propose the *minimum spanning pseudoforest problem*: Consider a graph G with n vertices and m edges. A pseudoforest *spans* G if it has the greatest possible number of edges. Assume every edge e has a real-valued *cost* $c(e)$. The cost of a set of edges is the sum of all its edge costs. A *minimum spanning pseudoforest* has the smallest cost possible. This paper presents an algorithm to find such a pseudoforest in time $O(m+n)$.

The pseudoforest problem relates to finding a minimum spanning tree. The best-known time for finding a minimum spanning tree is $O(m \log \beta(m,n))$ [GGST], where

$$\beta(m,n) = \min\{i \mid \log^{(i)} n \leq m/n\}.$$

Here log denotes logarithm base two, and $\log^{(i)} n$ is the i^{th} iterated logarithm, defined by $\log^{(0)} n = n$, $\log^{(i+1)} n = \log(\log^{(i)} n)$. Note that if $m/n \geq \log^{(i)} n$ for some constant i then $\beta(m,n) \leq i$, so the time to find a minimum spanning tree is $O(m)$. This paper presents a related result: If a graph has girth at least $\log^{(i)} n$ for some constant i then a minimum spanning tree can be found in $O(m)$ time.

Section 2 presents the results. This section closes with definitions and background from graph theory and data structures.

If S is a set and e an element, $S+e$ denotes $S \cup \{e\}$ and $S-e$ denotes $S-\{e\}$. For a graph G, $V(G)$ and $E(G)$ denote its vertex set and edge set, respectively. Hence for the given graph G, $n = |V(G)|$ and $m = |E(G)|$. An edge e is *incident* to a subgraph H if one or both ends is in $V(H)$ but $e \notin E(H)$.

A *tree* (*pseudotree*) *component* of a graph G is a connected component of G that is a tree (pseudotree). A spanning pseudoforest P for a graph G consists of every tree component of G, plus for every other connected component C of G, one or more pseudotree components that partition $V(C)$. Note that P contains exactly $|V(C)|$ edges of C.

The *set merging problem* [T] is to maintain a collection of disjoint sets which, after initialization, is subject to two operations:

unite(S, S')— form a new set $S \cup S'$, thereby destroying sets S and S';

find(e)— return the name of the set containing element e.

The set merging algorithm used in Section 2 is *union by size*: It represents each set S by a *union tree*, i.e., a tree whose nodes are the elements of S. A *unite* makes the root of the smaller union tree a child of the root of the larger. An operation *find*(v) is done by following the path in the union tree from v to the root. (No path compression is done). Hence a *unite* operation is $O(1)$ time and *find*(v) is $O(\log s)$, where s is the size of the set containing v.

In this paper a *priority queue* is a data structure on a universe that is partitioned into disjoint *queues*, where each element has a real-valued *cost*, and after initialization the following operations can be performed:

meld(Q, Q')— form a new queue by combining Q and Q', thereby destroying queues Q and Q';

find_min(Q)— return the smallest cost element in queue Q;

delete(e, Q)— remove element e from queue Q.

The algorithm used in Section 2 implements priority queues with Fibonacci heaps [FT]. The following time bounds hold: *meld* is $O(1)$; *find_min*(Q) is $O(\log s)$, where s is the size of Q; *delete*(e, Q) is $O(\log s)$, where s is the size of the Fibonacci tree containing e. Note these are amortized time bounds. Also to achieve the bound for *delete* the algorithm of [FT] is modified slightly, making it lazier: Unlike [FT] a queue does not keep track of its minimum element. Rather *find_min*(Q) links trees of Q until there is at most one tree of each rank, and then finds and returns the desired minimum. *delete*(e, Q) cuts e from its parent and adds the children of e to the list of trees of Q. The analysis of [FT] easily extends to prove the above time bounds. (The same time bounds can be achieved using binomial queues [Br] modified to do lazy melding).

2. The algorithm.

The algorithm is based on a locality property similar to one possessed by minimum spanning trees [T].

Lemma 2.1. Let P be a subgraph of a minimum spanning pseudoforest. Let e be a smallest cost edge incident to some tree component T of P. Then $P + e$ is a subgraph of a minimum spanning pseudoforest.

Proof. Let P^* be a minimum spanning pseudoforest containing P, and suppose P^* does not contain e. Let f be an edge of P^* that is incident to T such that the component of $P^* - f$ containing T is a tree (Specifically if T is in a tree component of P then f is an edge of P incident to T; if T is in a pseudotree component with cycle C, then f is an edge of P incident to T on C or on the path from T to C). By definition, $c(e) \leq c(f)$. Hence $P^* - f + e$ is the desired minimum spanning pseudoforest. ∎

The algorithm enlarges a subgraph P to a minimum spanning pseudoforest. For efficiency it grows the components of P at approximately the same rate. More precisely let $d(v)$ denote the degree of vertex v in the given graph G; the (*total*) *degree* of a subgraph H is $\sum\{d(v) | v \in V(H)\}$. The algorithm grows components so that they have similar degrees. The details are as follows.

The algorithm initializes P to contain every vertex v of G (v is initially a tree component of P). It then repeats the following step as long as P contains a tree component with an incident edge:

Enlarging Step. Choose a tree component T of smallest degree and add to P a minimum cost edge incident to T.

Correctness of this algorithm follows from the lemma; clearly pseudoforest P spans G when the algorithm halts.

The enlarging step is implemented with the following data structures. A set merging data structure maintains the partition of $V(G)$ induced by the components of P. Each component of P is marked as a tree or pseudotree. Each tree component T maintains its degree $d(T)$, and a priority queue of incident edges $Q(T)$, ordered by cost. An edge can be in two priority queues, in which case the two occurrences are linked by pointers. There is an array $C[1..2m]$, where $C[d]$ points to a doubly-linked list of all tree components of degree d with an incident edge.

With this data structure the enlarging step works as follows: The outermost loop examines the entries in C in increasing order to find the next smallest tree component T. T is removed from its C-list. The smallest edge e in $Q(T)$ is obtained using *find_min*. The set merging data structure *finds* the two components containing the ends of e, say T and S. If $S = T$ it is marked as a pseudotree. If $S \neq T$ then sets $V(S)$ and $V(T)$ are *united*; further if S is a tree it is deleted from its C-list, e is *deleted* from $Q(S)$ and $Q(T)$, these queues are *melded*, the new tree component $S \cup T$ gets degree $\delta = d(s) + d(t)$ and is added to the list $C[\delta]$ if its queue is nonempty. Finally in all cases, e is added to P.

To estimate the time, note that all initialization uses $O(m+n)$ time. The time for all enlarging steps, excluding priority queue *find_mins* and *deletes* and set merging *finds*, is $O(m+n)$. To estimate the time for *find_mins*, *deletes* and *finds*, define the *rank* of a component C as

$$r(C) = \lfloor \log d(C) \rfloor .$$

A simple induction shows that when T is chosen in the enlarging step, the size of any Fibonacci tree is at most $d(T)$ (recall that *find_min* is the only operation that enlarges Fibonacci trees; initially every edge is in its own Fibonacci tree). A similar induction shows that when T is chosen the height of the union tree for any component C is at most $\min\{r(C), 1 + r(T)\}$ (since T's height is at most $r(T)$). Thus the *find_min*, *find* and two *deletes* for T take time $O(\log d(T) + r(T)) = O(r(T))$. Let $\mathcal{T}(r)$ denote the set of all rank r tree components chosen as T in the enlarging step. Then the total *find_min*, *delete* and *find* time is at most a constant times

$$\sum_{r=0}^{\infty} r|\mathcal{T}(r)|.$$

For any rank r, any edge is counted in the degree of at most two trees of $\mathcal{T}(r)$ (since the enlarging step unites T into a pseudotree or increases the rank of the component containing T). Hence $\sum\{d(T) | T \in \mathcal{T}(r)\} \leq 2m$. Any $T \in \mathcal{T}(r)$ has $d(T) \geq 2^r$. Thus $|\mathcal{T}(r)| \leq m/2^{r-1}$. This implies the total time is at most a constant times $\sum_{r=0}^{\infty} rm/2^{r-1} = O(m)$.

Theorem 2.1. A minimum spanning pseudoforest can be found in time $O(m+n)$. ∎

Now we turn to the minimum spanning tree problem. Let P be a minimum spanning pseudoforest. Form a set C by choosing a maximum cost edge from each cycle of P.

Lemma 2.2. $P - C$ is a subgraph of a minimum spanning tree.

Proof. Let T be a minimum spanning tree with as many edges of P as possible. Suppose $P - C$ is not a subgraph of T. Let Q be a component of $(P - C) \cap T$ that is not a component of $P - C$; choose Q so it is not incident to an edge of C. Let e be an edge of P incident to Q such that the component of $P - e$ containing Q is a tree (e is found as in Lemma 2.1). Let f be an edge incident to Q in the fundamental cycle of e in T (f exists since $e \notin T \cup C$). Then $P - e + f$ is a spanning pseudoforest, whence $c(e) \leq c(f)$. $T - f + e$ is a spanning tree containing more edges of P than T, whence $c(f) < c(e)$. This contradiction proves the lemma. ∎

The lemma justifies the following minimum spanning tree algorithm. Find a minimum spanning pseudoforest P. Form the forest F by deleting a maximum cost edge from each cycle of P; form the graph G' by contracting each tree of F to a vertex. Find a minimum spanning tree T of G'. Now $T \cup F$ is a minimum spanning tree of G.

This algorithm improves the bound for minimum spanning trees in the following special case. For the improvement it suffices to find T using the minimum spanning tree algorithm of [FT], which uses time $O(m\beta(m,n))$ but is slightly simpler than [GGST]. Recall the *girth* g of a graph is the length of a shortest cycle [H].

Theorem 2.2. Let G be a graph with girth $g \geq \log^{(i)} n$ for some constant i. Then a minimum spanning tree of G can be found in time $O(m)$.

Proof. Except for finding T, the algorithm uses linear time. Let $n' = |V(G')|$, $m' = |E(G')|$, so T is found in time $O(m'\beta(m',n'))$. Clearly $n' \leq n/g$ and $m' \leq m$. Note that $m\beta(m,n)$ is an increasing function of m (since $\beta(m,n) \leq n$ and $\beta(m+1,n) \geq \beta(m,n) - 1$). Hence $m'\beta(m',n') \leq m\beta(m,n') \leq m\beta(m,n/g)$. Since $m \geq n$, $\beta(m,n/g) \leq \beta(n,n/g) \leq \beta(ng,n)$. Since $g \geq \log^{(i)} n$, $\beta(ng,n) \leq i$ by definition. This gives the theorem. ∎

In conclusion, a minimum spanning tree can be found in linear time if the graph has density or girth at least $\log^{(i)} n$. This narrows the open case down to graphs that are extremely sparse.

Acknowledgments.

We thank David Shmoys for pointing out the applications of pseudoforests to networks.

References.

[Be] C. Berge, *Graphs*, 2^{nd} revised edition, North-Holland, New York, 1985.

[Br] M. R. Brown, "Implementation and analysis of binomial queue algorithms", *SIAM J. Comput. 7*, 3, 1978, pp. 298-319.

[D] G.B. Dantzig, *Linear Programming and Extensions*, Princeton Univ. Press, Princeton, N.J., 1963.

[FT] M.L. Fredman and R.E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms", *Proc. 25th Annual Symp. on Found. of Comp. Sci.*, 1984, pp.338-346; also *J. ACM*, to appear.

[GGST] H.N. Gabow, Z. Galil, T.H. Spencer and R.E. Tarjan, "Efficient algorithms for finding minimum spanning trees in undirected and directed graphs", *Combinatorica 6*, 2, 1986, pp. 109-122.

[GPS] A. V. Goldberg, S.A. Plotkin, G.E.Shannon, "Parallel symmetry-breaking in sparse graphs", *Proc.* 19^{th} *Annual ACM Symp. on Theory of Comp.*, 1987, pp. 315-324.

[H] F. Harary, *Graph Theory*, Addison-Wesley, Reading, Mass., 1969.

[HK] M. Held and R. M. Karp, "The traveling-salesman problem and minimum spanning trees", *Oper. Res. 18*, 1970, pp. 1138-1162.

[L] E.L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[MH] S. Mitchell and S. Hedetniemi, "Linear algorithms for edge-coloring trees and unicyclic graphs", *Inf. Proc. Letters, 9*, 3, 1979, pp. 110-112.

[PQ] J.-C. Picard and M. Queyranne, "A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory", *Networks, 12*, 1982, pp. 141-159.

[T] R.E.Tarjan, *Data Structures and Network Algorithms*, SIAM Monograph, Philadelphia, Pa., 1983.

[W] H.H. Westermann, "Efficient algorithms for matroid sums", Ph. D. Dissertation, Dept. of Comp. Sci., Univ. of Colorado, in preparation.

[WW] N. White and W. Whiteley, "The algebraic geometry of motions of bar-and-body frameworks", *SIAM J. Alg. Disc. Meth. 8*, 1, 1987, pp. 1-32.