Acta Informatica 23, 223-229 (1986)

A Faster Approximation Algorithm for the Steiner Problem in Graphs

Y.F. Wu[1,3], P. Widmayer[2,4], and C.K. Wong[2]

[1] Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201, USA

[2] IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, USA

Summary. We present an algorithm for finding a Steiner tree for a connected, undirected distance graph with a specified subset S of the set of vertices V. The set $V-S$ is traditionally denoted as Steiner vertices. The total distance on all edges of this Steiner tree is at most $2(1-1/l)$ times that of a Steiner minimal tree, where l is the minimum number of leaves in any Steiner minimal tree for the given graph. The algorithm runs in $O(|E| \log |V|)$ time in the worst case, where E is the set of all edges and V the set of all vertices in the graph. It improves dramatically on the best previously known bound of $O(|S|\,|V|^2)$, unless the graph is very dense and most vertices are Steiner vertices. The essence of our algorithm is to find a generalized minimum spanning tree of a graph in one coherent phase as opposed to the previous multiple steps approach.

1. Introduction

Consider a connected, undirected distance graph $G=(V, E, d)$ and a set $S \subseteq V$, where V is the set of vertices in G, E is the set of edges in G, and d is a distance function which maps E into the set of nonnegative numbers. A *path* in G is a sequence of vertices $v_1, v_2, \ldots, v_k$ of V, such that for all i, $1 \leqq i < k$, $(v_i, v_{i+1}) \in E$ is an edge of the path. For a path p, $V(p)$ is the set of all vertices, and $E(p)$ the set of all edges in p. A *loop* is a path $v_1, \ldots, v_k$ with $v_1 = v_k$. The *length* of a path is the sum of the distances of its edges. A connected subgraph $G_s=(V_s, E_s, d_s)$ of G with $S \subseteq V_s \subseteq V$, $E_s \subseteq \{(v_1, v_2) | (v_1, v_2) \in E, \{v_1, v_2\} \subseteq V_s\}$, and d_s equals d, restricted to E_s, is called a *Steiner tree* for G and S, if G_s does not contain a loop. Given a Steiner tree for G and S, $G_s=(V_s, E_s, d_s)$, $D(G_s)$ is

[3] The work of this author was partially supported by the National Science Foundation under Grants MCS 8342682 and ECS 8340031. This work was performed while this author was a summer visitor at the IBM T.J. Watson Research Center. Current address: MCC CAD 9430 Research Blod., Austin, TX 78759, USA

[4] On leave from: Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, Postfach 6380, D-7500 Karlsruhe, Federal Republic of Germany

defined as $\sum_{e \in E_s} d_s(e)$, and is called the *total distance* of G_s. A Steiner tree G_s for G and S is called a *Steiner minimal tree*, if its total distance is minimal among all Steiner trees for G and S. This minimal distance is called $D_{\min}(G)$. Note that vertices in S are required to be in any Steiner tree for G and S. On the other hand, vertices in $V-S$, which are traditionally called *Steiner vertices*, are not required to be in a Steiner tree, but may be used to achieve a small total distance. There have been confusions on whether to call the set S or the set $V-S$ Steiner vertices. In this paper we adhere to the definition used in the fundamental paper of Gilbert and Pollak [7]. The same terminology is used in a number of later papers [2, 3, 5, 11].

The problem of finding a Steiner minimal tree for given G and S has been shown to be NP-complete [8], even for a restricted class of distance functions [6]. Therefore, we are interested in finding a Steiner tree with total distance close to the total distance of a Steiner minimal tree. Algorithms have been proposed for this task in the literature [9, 12]. [12] presented an algorithm for finding a Steiner tree G' with $D(G')/D_{\min}(G) \leqq 2(1-1/|S|)$, whereas [9] described a procedure for finding a Steiner tree G'' with $D(G'')/D_{\min}(G) \leqq 2(1-1/l)$, and l is the minimum number of leaves in any Steiner minimal tree for G and S. The runtime of both algorithms is proportional to $|S|\,|V|^2$. Note that $l \leqq |S|$. Thus, the bound is expressed sharper in [9] than in [12], even though the underlying arguments are identical.

We will describe an algorithm for computing a Steiner tree using the same strategy as in [9], hence with the same bound on its total distance, but in time bounded by $O(|E| \log |V|)$. This speedup is achieved by the combination of a process for finding shortest paths, similar to [4], with a process for constructing a minimum spanning tree (MST), similar to [10].

2. An Algorithm for Approximating a Steiner Minimal Tree

Let $G=(V, E, d)$ be a given connected, undirected distance graph, and $S \subseteq V$ the set of vertices for which a Steiner tree is desired. Our approach is to use *a generalized minimum spanning tree of a graph* to approximate the Steiner minimal tree. Our algorithm is in line with Algorithm H in [9]. The major differences are: (a) we perform the task in one step, and (b) our time bound is superior in most cases. For the sake of completeness, we outline Algorithm H of [9] with some notations rephrased in the following:

Algorithm H: Steiner Tree [9]

1. Construct the complete distance graph $G_1=(V_1, E_1, d_1)$, where $V_1=S$ and, for every $(v_i, v_j) \in E_1, d_1(v_i, v_j)$ is equal to the distance of a shortest path from v_i to v_j in G.
2. Find a minimum spanning tree G_2 of G_1.
3. Construct a subgraph G_3 of G by replacing each edge in G_2 by its corresponding shortest path in G. (If there are several shortest paths, pick an arbitrary one.)

4. Find a minimum spanning tree G_4 of G_3.
5. Construct a Steiner tree G_5 from G_4 by deleting edges in G_4, if necessary, so that no leaves in G_5 are Steiner vertices.

We will define and describe an algorithm for computing a generalized minimum spanning tree G_s of a graph G and a specified set of vertices $S \subseteq V$ in time $O(|E| \log |V|)$ in the next two sections. G_s and G_2 both span the set S. They differ in that G_s uses edges in E directly while G_2 uses edges in E_1. Our algorithm constructs a G_s with the following properties: (a) during the construction of G_s, G_1 is not explicitly constructed; (b) G_s has all the path information of edges of G_2, as conveyed in G_3; (c) G_s is a tree, thus is equivalent to G_4; (d) no leaves in G_s are Steiner vertices. Therefore, we can conclude that G_s is, in general, equivalent to G_5 in Algorithm H although there are instances in which they might have different total distance. The cause of the possible difference is that Step 4 of Algorithm H may make decisions on deleting edges in G_3 which result in a better or worse final Steiner tree, depending on occasions. The total distance of G_s is at most $2(1-1/l)$ times of that of a Steiner minimal tree, where l is the number of leaves in a Steiner minimal tree. This bound is the same as that of G_5, the result of Algorithm H. In fact, we can substitute G_s for T_H, and D_s for D_H in Theorem 1 and Theorem 2 and their proofs in [9]. This yields the following theorem.

Theorem. *For a connected, undirected distance graph $G=(V, E, d)$, and a set of vertices $S \subseteq V$, a Steiner tree G_s for G and S with total distance at most $2(1-1/l)$ times that of a Steiner minimal tree for G and S can be computed in $O(|E| \log |V|)$time.* □

3. Finding a Generalized Minimum Spanning Tree of a Graph

Using the above notion of G_1 and G_2, were define the following.

Definition. A generalized minimum spanning tree $G_s(V_s, E_s, d_s)$ of a given connected, undirected distance graph $G=(V, E, d)$ and a set of vertices $S \subseteq V$, denoting vertices in $V-S$ as Steiner vertices, is a tree subgraph of G such that (a) there exists a minimum spanning tree $G_2(V_2, E_2, d_2)$ of $G_1(V_1, E_1, d_1)$ such that $\forall e=(v_i, v_j) \in E_2$: the unique path in G_s from v_i to v_j is of length $d_2(e)$; (b) all leaves in G_s are in S. □

In other words, G_s is the actual realization of G_2 on the graph G. It is clear that G_s is a Steiner tree, and a good approximation of the Steiner minimal tree. To make our presentation simpler, we concentrate on finding the G_2 associated with G_s in the following algorithm. After the main idea is clarified, we will then modify the algorithm so that G_s is computed. For each vertex v in V, we use *source*(v) to represent a vertex in S which is closest to v and *length*(v) to represent the distance from *source*(v) to v. If there is a tie among points in S, then *source*(v) is picked arbitrarily.

Following the principle of Kruskal's algorithm, we treat the vertices in S in the beginning as a forest of $|S|$ separate trees and try to merge them into a

single tree. A priority queue Q is used to store frontier vertices of paths extended from the trees and possible edges to be used for linking the trees. Tuples in Q have the format (t, d, s) where $t \in V$, $s \in S$ and d is the length of a path from s to t. If $t \in S$ then (s, t) is a possible edge to be used in a generalized minimum spanning tree, otherwise ($t \in V-S$) s is a possible candidate for *source*(t). The order of entries in the priority queue is determined by non-decreasing values of d's in the tuples. We use an algorithm in [1] to efficiently implement INSERT and FIND-DELETE-MIN opertions for the priority queue.

An efficient UNION-FIND algorithm, for instance as described in [1], is used to quickly determine if two vertices in S belong to the same tree in the forest, and merge two trees in the forest into one tree.

Algorithm M: Generalized Minimum Spanning Tree of a Graph

1. For all $q \in S$, *source*(q) $\leftarrow q$ and *length*(q) $\leftarrow 0$.
   For all $q \in V-S$, *source*(q) $\leftarrow$ *undefined* and *length*(q) $\leftarrow \infty$.

2. For all $q \in S$, put (r, d, q) into the priority queue Q where $(q, r) \in E$ and $d = d((q, r))$. We use an assumed order on the vertices in S to ensure putting only one of $(r, d((r, q)), q)$ and $(q, d((q, r)), r)$ into Q, if both $q, r \in S$.

3. Partition the vertices in S into $|S|$ sets (trees), such that each set contains one vertex.

4. *While* not all vertices in S are in a single set, *do*:
   Choose a tuple (t, d, s) with minimum d in Q and remove it from Q.

   Case 1. *source*(t) is undefined: *source*(t) $\leftarrow s$, *length*(t) $\leftarrow d$. For all r such that $(t, r) \in E$ and *source*(r) is undefined, put $(r, d((t, r)) + d, s)$ into Q.

   Case 2. *source*(t) and s are in the same set: Do nothing.

   Case 3. *source*(t) and s are not in the same set:

   Case 3.1. $t \in S$: Merge the two sets (trees) containing s and t into one set and record that there is an MST edge (which is a path in G) between s and t.

   Case 3.2. $t \in V-S$: Put (*source*(t), d + *length*(t), s) into Q.

Based on the correctness of Kruskal's algorithm, the only thing we need to establish for the correctness of Algorithm M is that each time we merge two trees S_1 and S_2 into one tree, they are indeed a closest pair of trees. Assume the contrary, i.e., there is another pair of subtrees S_3 and S_4 which are closer, and are closest among all pairs. But according to Algorithm M, the paths reaching out from S_3 and S_4 should have touched each other because the two paths have length less than the distance between S_1 and S_2 (Case 3.2 of Step 4) so that a path from a vertex of S_3 to a vertex of S_4 (or vice versa) should have been put into the priority queue. The path should also have been retrieved later from the priority queue (Case 3.1 of Step 4) and used to link S_3 and S_4. This is a contradiction. Therefore, Algorithm M constructs a correct generalized minimum spanning tree.

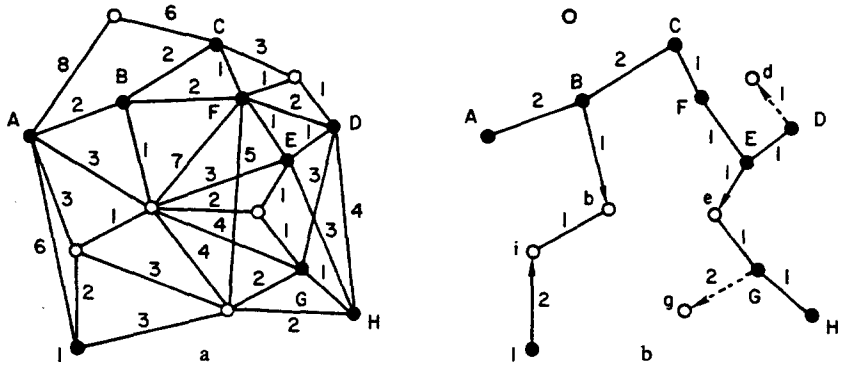

Fig. 1. An example

Step 1 of Algorithm M can be carried out in $O(|V|)$ time; Step 2 can be done in $O(|E| \log |V|)$ time; Step 3 runs in $O(|S|)$ time. Let us have a closer look at Step 4. Note that a tuple in Q is generated either by extending a path by an edge in E as in Case 1 of Step 4 or by trying to connect two vertices in S as in Case 3.2 of Step 4. In either case, an edge is used in the extension or connection. Furthermore, each edge can be used only once, in one of these two cases. Thus the total number of tuples ever generated (including those in the initialization process) in the algorithm is $O(|E|)$. Since each tuple is processed in $O(\log |V|)$ time in the priority queue operations and UNION-FIND operations, Step 4 can be carried out in $O(|E| \log |V|)$ time. Therefore the total time spent by the algorithm is $O(|E| \log |V|)$.

Figure 1a shows an example graph where vertices in S are solid dots. Figure 1b shows a generalized minimum spanning tree obtained by our algorithm where arrows represent the direction of growing the edges from vertices in S, and lower case labels represent nodes whose source nodes have the corresponding upper case labels. Generalized minimum spanning tree edges are not explicitly represented in the graph, because we do not use them in the Steiner tree construction process. Dotted edges are edges that are not used in the resulting generalized minimum spanning tree.

3. Remarks

Note that we can modify Algorithm M for the generalized MST construction to deliver an explicit description for the path corresponding to each generalized MST edge. In so doing, the modified algorithm computes G_s directly.

(1) For each vertex v in V, introduce a new attribute $pred(v)$ which indicates its immediate predecessor on the path from $source(v)$ to v. If $v \in S$, then $pred(v)$ is undefined. In Step 1 of Algorithm M, all predecessors are initialized as undefined.

(2) Modify the format of tuples stored in Q as (t, d, s, p_1, p_2) where $t, p_1, p_2 \in V$, $s \in S$ and the interpretation of the tuples depends on whether $t \in S$ or not.

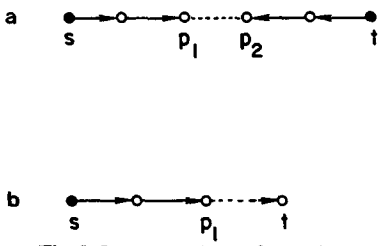

Fig. 2. Interpretations of a tuple in Q

Case 1. $t \in S$: There is an edge in E between p_1 and p_2 and $source(p_1)=s$ and $source(p_2)=t$. $d=length(p_1)+length(p_2)+d((p_1,p_2))$. Thus (s,t) is a possible edge to be used in the generalized minimum spanning tree (see Fig. 2a).

Case 2. $t \in V-S$: There is an edge in E between p_1 and t. The source of p_1 has been determined to be s and $d=length(p_1)+d((p_1,t))$. Thus s is a possible candidate for $source(t)$. p_2 is undefined (see Fig. 2b).

The initialization in Step 2 must also be modified accordingly.

(3) In Step 4, get a tuple (t,d,s,p_1,p_2) from Q. In Case 1, put $(r,d((t,r))+d,s,t,$ *undefined*) into Q instead of $(r,d((t,r))+d,s)$. In Case 3.1, record (p_1,p_2) as a pair of header nodes for edge (s,t) in the generalized minimum spanning tree so that the actual path can be traced out by following the *pred* links of the header nodes to their corresponding source nodes. In Case 3.2, put $(source(t), d+length(t), s, p_1, t)$ into Q instead of $(source(t), d+length(t), s)$.

As to the complexity of Algorithm M (also the modified algorithm), the time bound of $O(|E|\log|V|)$ in general is better than the $O(|S|\,|V|^2)$ one in [9]. One way of viewing the difference is that when $|S|$ is relatively large compared to $\log|V|$, the algorithm in [9] has to perform $|S|$ shortest-path-finding processes, while our algorithm calculates the necessary shortest paths simultaneously with the construction of the generalized minimum spanning tree. Another case in which our algorithm performs better is when G is not a very dense graph, so that $|E|$ is less than $O(|V|^2/\log|V|)$.

Another point of interest is that the total number of INSERT and the total number of FIND-DELETE-MIN priority queue operations in Algorithm M can both be simultaneously $\Omega(|E|)$ even when $|E|=\Theta(|V|^2)$. The following is an example. Consider graph $G=(V,E,d)$ with $S \subseteq V$, and the set of Steiner vertices $S'=V-S$. Let

(1) $S=\{s_1,s_2,\ldots,s_m\}$, and let

(2) $S'=\{s'_1,s'_2,\ldots,s'_n\}$, and let

(3) $E=\{(s_i,s'_j)\,|\,1\leqq i\leqq m,\ 1\leqq j\leqq n\}$. That is, G is a complete bipartite graph, w.r.t. S and S'. Let d_{ij} denote the distance of edge (s_i,s'_j). Let the distance be such that

(4) $d_{ij}<d_{ik}$ whenever $j<k$, and

(5) $d_{in}<d_{jl}$ whenever $i<j$, and

(6) $d_{ij}+d_{kj}>d_{pq}$ for all i, j, k, p, q. In other words, the distances in ascending order are $d_{11}, d_{12}, \ldots, d_{1n}, d_{21}, \ldots, d_{2n}, \ldots, d_{m1}, \ldots, d_{mn}$. Then, (6) is equivalent to saying that

(7) $d_{11}+d_{21}>d_{mn}$ when (4) and (5) are taken into account. A set of distance values fulfilling these conditions is, for instance,

(8) $d_{ij}=mn+(i-1)n+(j-1)$. So, recall that we consider the graph G with definitions (1), (2), (3), and (8). Apply Algorithm M to this graph. Then Step 2 of the algorithm will put an entry (s'_j, d_{ij}, s_i) into Q, for each $1 \leqq i \leqq m$, $1 \leqq j \leqq n$, altogether $mn=|E|$ entries. Before the first time Case 3.1 in Step 4 occurs when retrieving an entry from Q, condition (6) requires that all entries entered by Step 2 have been removed from Q. Hence, there are at least $mn=|E|$ INSERT and FIND-DELETE-MIN operations carried out on Q, with $|E|$ of higher order than $|V|$. For $m=n$, e.g., $|E|=\Theta(|V|^2)$. Thus simply trying to use alternative priority queue techniques such as the Fibonacci heaps to speed up the priority queue operations will not improve the given time bound of Algorithm M.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Reading: Addison-Wesley 1974
2. Aneja, Y.P.: An Integer Linear Programming Approach to the Steiner Problem in Graphs. Networks **10**, 167 (1980)
3. Beasley, J.E.: An Algorithm for the Steiner Problem in Graphs. Networks, **14**, 148 (1984)
4. Dijkstra, E.W.: A note on two problems in connection with graphs. Numer. Math. **1**, 269-271 (1959)
5. Du, D.Z., Yao, E.Y., Hwang, F.K.: A short proof of a result of Pollak on Steiner minimal trees. J. Comb. Theory, Ser. A, **32**, 396 (1982)
6. Garey, M.R., Graham, R.L., Johnson, D.S.: Some NP-complete Geometric Problems. 8th Annual ACM Symposium on Theory of Computing, pp. 10-22, 1976
7. Gilbert, E.N., Pollak, H.U.: Steiner Minimal Trees. SIAM J. Appl. Math. **16**, 1 (1968)
8. Karp, R.M.: Reducibility among Combinatorial Problems. In: Complexity of Computer Computations, pp. 85-103. New York: Plenum Press 1972
9. Kou, L., Markowsky, G., Berman, L.: A Fast Algorithm for Steiner Trees. Acta Inf. **15**, 141-145 (1981)
10. Kruskal, J.B., Jr.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Am. Math. Soc. **7**, 48-50 (1956)
11. Pollak, H.U.: Some remarks on the Steiner Problem. J. Comb. Theory, Ser. A, **24**, 278 (1982)
12. Takahashi, H., Matsuyama, A.: An Approximate Solution for the Steiner Problem in Graphs. Math. Jap. **24**, 573-577 (1980)

Received September 25, 1984/September 14, 1985