Finding the k Smallest Spanning Trees

David Eppstein
Department of Information and Computer Science
University of California, Irvine, CA 92717

Abstract

We give improved solutions for the problem of generating the k smallest spanning trees in a graph and in the plane. Our algorithm for general graphs takes time $O(m \log \beta(m,n) + k^2)$; for planar graphs this bound can be improved to $O(n + k^2)$. We also show that the k best spanning trees for a set of points in the plane can be computed in time $O(\min(k^2 n + n \log n, k^2 + kn \log(n/k)))$. The k best orthogonal spanning trees in the plane can be found in time $O(n \log n + kn \log\log(n/k) + k^2)$.

1 Introduction

One of the fundamental problems in graph theory is the computation of a *minimum spanning tree* (MST). Given an undirected graph G with weights on each edge, the MST of G is the tree spanning G having the minimum total edge weight among all possible spanning trees. This problem has been intensely studied, and good algorithms are known; currently the best known bound, by Gabow et al. [16], is $O(m \log \beta(m,n))$ for a graph with n vertices and m edges. Here $\beta(m,n)$ is defined to be $\min\{i | \log^{(i)} n \le m/n\}$, and $\log^{(i)} x$ denotes the log function iterated i times. This is extremely close to linear time. For planar graphs, a MST can be found in linear time [8].

Minimum spanning trees have applications in many areas, including network design, VLSI, and geometric optimization. Yet in many cases what is desired is not necessarily the best spanning tree, but rather a "good" tree with some other properties that may be difficult to quantify. For instance, minimum spanning trees can be used to approximate a Euclidean travelling salesman tour, but it might be the case that some tree other than the minimum could yield a better tour. An approach to this difficulty is to generate a number of "good" trees, and then choose among them by whatever other criteria are desired.

A natural formulation of this problem is to find the k least weight spanning trees, for some input parameter k. This problem is not so well known as the usual MST problem, but it has been previously studied. It should be contrasted with the much harder problem of finding the k best possible spanning tree weights [19].

The main previous result, by Katoh et al [18], is that the k best spanning trees can be found in time $O(m \log \beta(m,n) + km)$. This result improved several prior results by Burns and Haff [6], Camerini et al [7], and Gabow [15]. Apparently, Dov Harel has discovered an $O(m \log \beta(m,n) + kn \log^2 n)$ time algorithm [13]; this is an improvement for graphs that are not too sparse.

Frederickson [13] considered the problem for small k, in particular $k = O(\sqrt{m})$. His algorithm uses a technique for maintaining a MST in a dynamically changing graph, and runs in time $O(m \log \beta(m,n) + k^2\sqrt{m})$. Frederickson also gave a version of his algorithm for planar graphs that runs in time $O(n + k^2 \log^2 n)$. Recently, Frederickson's technique for maintaining a MST in a dynamic planar graph has been improved [12, 17], leading to a bound for the k best spanning trees problem of $O(n + k^2 \log n)$; this improves the bound of [18] whenever $k = O(n/\log n)$.

Another well known MST problem is that of finding the MST of a set of points in the plane, with the weight of an edge connecting points x and y being the Euclidean distance between the two. This can be solved in time $O(n \log n)$, using the fact that all MST edges must appear in the Delauney triangulation of the points [3]. Strangely, the problem of finding the k best minimum spanning trees of a set of points in the plane does not seem to have been studied; however it can clearly be solved in time $O(kn^2)$ using the technique of [18] on the complete graph of all pairs of points.

In this paper we again consider the problem of generating the k best spanning trees, both in a graph and in the plane, again for small k. We show that the k best spanning trees in any graph can be found in time $O(m \log \beta(m,n) + k^2)$; this is better than the previous $O(m \log \beta(m,n) + k^2\sqrt{m})$ bound. For planar graphs, our algorithm can be made to run in time $O(n + k^2)$, improving the previous $O(n + k^2 \log n)$ bound. Both algorithms improve the $O(m \log \beta(m,n) + km)$ bound of [18] whenever $k = O(m)$.

The general graph approach clearly leads to an $O(n^2 + k^2)$ bound for the Euclidean k best spanning trees problem, which improves the previous $O(kn^2)$ bound when $k = O(n^2)$. We describe two modifications to this technique. The first finds the k best spanning trees in time $O(k^2 n + n \log n)$; it is best when $k = O(\log n)$, and is $O(n \log n)$ for $k = O(\sqrt{\log n})$. The second achieves time $O(k^2 + kn \log(n/k))$, and is better than $O(n^2 + k^2)$ when $k = O(n)$. We also consider alternate planar metrics; in particular we give an $O(k^2 + kn \log\log(n/k) + n \log n)$ algorithm to find the k best orthogonal spanning trees.

The intuition behind our algorithms is as follows. Each of the k best trees (other than the MST) differs from a better tree by the deletion of a single edge and its replacement by another edge. Therefore in all the k best trees there are $O(k)$ edges which are added to or removed from the MST. If we can quickly identify a superset of these edges, we can reduce the original k best spanning trees problem to a new problem the size of the superset; we can solve this reduced problem with previously known algorithms.

In more detail, the algorithm for finding spanning trees in graphs consists of three stages. First, we show that many edges of the MST must be contained in every one of the k best spanning trees; therefore, we can contract the input graph G to a new graph G' having only $O(k)$ vertices. Then, we show that many edges not in the MST can also not be in any of the k best spanning trees; therefore, we can remove them from the graph, leaving only $O(k)$ edges. Finally, we apply the algorithm of [18], which takes time $O(k^2)$ on the reduced graph. Our algorithms for geometric spanning trees similarly rely on reducing the problem to a small graph, but are more complicated.

We have recently used the idea of removing and contracting edges in algorithms for the related problem of maintaining a minimum spanning tree in a changing graph, subject to an offline sequence of edge insertions and deletions, and for a similar dynamic geometric minimum spanning tree problem [10]. A recent paper of Frederickson [14] improves the k smallest spanning trees algorithm of [18], and again uses the reduction described here to

improve some of our time bounds as well.

Throughout this paper we allow graphs to have multiple edges between the same pair of vertices. Therefore we do not denote an edge by its adjacent vertices, but rather treat it as a separate entity. The presence of multiple adjacencies between two vertices cannot affect the MST of a graph, but may affect the outcome of the k best spanning trees computation.

2 Contracting Required Edges

Given a graph G, and an edge e connecting vertices x and y in G, we define the *contraction* $G \cdot e$ to be the graph G' having as vertices $V(G) - y$, and edges $c_e(E(G))$, where the function c_e throws away edges connecting x and y, substitutes for each edge e' connecting y and any vertex $z \neq x$ a new edge e'' connecting x and z, and leaves unchanged all remaining edges. The contraction function c_e is defined to preserve the weight $w(e)$ of each edge not thrown away.

Lemma 1. *For any edge e in a spanning tree T of graph G, $c_e(T)$ is the MST of $G \cdot e$ if and only if T is the least weight spanning tree of G containing e.*

Proof: Obvious from the definitions. □

The following characterization of MSTs is well known:

Lemma 2. (folklore) *Let v be any vertex of G, and e be the least weight edge adjacent to v. Let T be the MST of $G \cdot e$. Then $T + e$ is the MST of G.*

Now for any edge e in the spanning tree, disconnecting the tree into two components T_1 and T_2, define the *replacement edge* $r_G(e)$ to be the least weight edge in G, other than e, between a vertex in T_1 and one in T_2.

Lemma 3. (folklore) *For any edge e in the MST T of a graph G, such that $G - e$ is connected, $T - e + r_G(e)$ is the MST of $G - e$.*

Proof: If G has only two vertices, the lemma is obvious. Otherwise, find some edge $f \neq e$ that is a leaf in the MST T of G, and let the leaf vertex adjacent to f be x. Then by lemma 2, f is the least weight edge adjacent to x, or it would not be in T, and by lemma 1 $c_f(T)$ is the MST of $G \cdot f$. Also, contracting f does not change the components T_1 and T_2 formed by removing e, and therefore $r_{G \cdot f}(e) = r_G(e)$. By induction $c_f(T) - e + r_G(e)$ is the MST of $(G \cdot f) - e = (G - e) \cdot f$, and using lemma 1 again gives that $T - e + r_G(e)$ is the least weight spanning tree of $G - e$ containing f. But by lemma 2, this must be the MST of $G - e$. □

Lemma 4. *[21] Given a MST T of a graph G, the replacements $r_G(e)$ for all edges e in T can be computed in time $O(m\alpha(m, n))$.*

This bound is never worse than the $O(m \log \beta(m, n))$ time required for constructing the MST of G. The following version of lemma 4 is given in [5].

Lemma 5. *Given a MST T of a planar graph G, the replacements $r_G(e)$ for all edges e in T can be computed in time $O(n)$.*

Lemma 6. *Given a MST T of a graph G, and the values of r_G for each tree edge, in linear time we can compute a set S of $n-k$ tree edges that must be contained in all of the k best spanning trees for G.*

Proof: For each edge e, let $w'(e) = w(r_G(e)) - w(e)$. In other words, w' is the extra weight we have to add to the tree if we remove e from the graph. The values of w' can be computed as above. Then find the edges with the $(k-1)$ smallest values of w', using a linear time selection algorithm [4]. Let S be all the remaining edges.

Then for each edge e in S there are at least $k-1$ edges e' with $w(T - e' + r_G(e')) \leq w(T - e + r_G(e))$, and therefore together with T there are at least k trees better than $w(T - e + r_G(e))$, which is in turn (by lemma 3) better than any other spanning tree not containing e. Therefore all the k best spanning trees must contain e. □

Lemma 7. *Given a MST T of a graph G, in time $O(m\alpha(m,n))$ we can find a set of edges S and a graph G' having only k vertices, such that the k best spanning trees of G are exactly the k best spanning trees of G' together with the edges of S. In a planar graph this can be performed in linear time.*

Proof: Let S be the set constructed in lemma 6, and let $G' = G \cdot S$ be the graph formed by contracting each of the edges in S. The contraction can easily be performed in linear time. □

3 Subtracting Useless Edges

Just as each tree edge has a single non-tree replacement edge $r_G(e)$, it turns out that each non-tree edge has a single tree replacement edge. In particular, define $R_G(e)$ for an edge e connecting vertices x and y to be that edge on the tree path between x and y having the highest weight.

Lemma 8. *If T is the MST of a graph G, then $c_e(T - R_G(e))$ is the MST of $G \cdot e$.*

Proof: We contract tree edges other than $R_G(e)$ one at a time, using lemma 1 to show that they must belong to both MSTs. □

As before, R_G can be computed efficiently for all edges:

Lemma 9. *[21] The replacement edges $R_G(e)$ for each edge e in a graph G, given a MST of G, can be computed in time $O(m\alpha(m,n))$.*

Lemma 10. *The replacement edges $R_G(e)$ for each edge e in a planar graph G can be computed in linear time.*

Proof: Since the minimum spanning tree of a planar graph is the complement of the maximum spanning tree of the dual [12], this follows from lemma 5. □

And as before, this lets us simplify the graph:

Lemma 11. *Given a graph G and its MST T, we can find a set S' of $m - n - k$ non-tree edges such that none of the k best spanning trees contains any edge in S', in time $O(m\alpha(m, n))$.*

Proof: Define $W(e)$ to be $w(e) - w(R_G(e))$; then $W(e)$ is the weight added by including e in a spanning tree. Let f be the non-tree edge having the $(k-1)$st smallest value of W; as before f can be found by a linear time selection algorithm. Then as before, for any e with $W(e) > W(f)$, there are at least k trees better than $T - R_G(e) + e$, and therefore also better than any other tree containing e. Therefore e can never be included in any of the k best spanning trees for G. □

Putting it all together, we have our result:

Theorem 1. *The k least weight spanning trees of a graph can be found in time bounded by $O(m \log \beta(m, n) + k^2)$; for a planar graph they can be found in time $O(n + k^2)$.*

Proof: By results of [16] and [8], the MST of the graph can be found in the given bounds. We can use lemma 7 to reduce the graph to one in which there are k vertices, and therefore $k - 1$ tree edges. Then we can use lemma 11 to reduce the graph to one in which there are $k - 1$ non-tree edges. Therefore the total number of edges in the final reduced graph is $O(k)$. Applying the algorithm of [18] gives the desired total bound. □

4 Euclidean Spanning Trees

We now consider the Euclidean k best spanning trees problem. More precisely, given a set of n points in the plane, we can imagine forming the complete graph on these points, with the weight of an edge equal to the distance between the corresponding vertices.

Theorem 2. *The k best spanning trees for a set of points in the plane can be found in time $O(n^2 + k^2)$.*

Proof: Simply form the graph described above, and use the algorithm of the previous sections. □

To do better than this, we will need to remove many edges quickly from the complete graph, without explicitly constructing the graph. To do this, we will use the standard geometric technique of Voronoi diagrams.

The *order r Voronoi diagram* is a subdivision of the plane into regions. Each region can be denoted $v(S)$ for some set S containing exactly r of the input points (which in this context are called *sites*), and is the locus of points x for which all sites in S are closer to x than any site not in S. A set S corresponds to a Voronoi region $v(S)$ exactly when there is a circle in the plane containing S and no other of the sites; then the center of the circle is one of the points in $v(S)$.

Lemma 12. *If (x, y) is an edge in one of the k best spanning trees, then x and y are both sites in some set S corresponding to a Voronoi region $v(S)$ in the order $(k + 1)$ Voronoi diagram of the input points.*

Proof: Let x and y be two arbitrary input points. Let e in the MST T of the points be $R_G((x,y))$, and let the removal of e disconnect T into two components T_1 and T_2. Without loss of generality $x \in T_1$ and $y \in T_2$.

Consider the circle C for which line segment xy is diameter. For each site z other than x and y in this circle, if z is in T_2 then (x,z) connects T_1 and T_2, and is shorter than (x,y); therefore $T-e+(x,z)$ is a better spanning tree than $T-e+(x,y)$. Similarly, if z is in T_1 then $T-e+(z,y)$ is better than $T-e+(x,y)$. But $T-e+(x,y)$ is the best spanning tree containing (x,y); therefore if (x,y) is in any of the k best spanning trees, there can be at most $k-1$ possible sites z, and C contains at most $k+1$ sites including x and y. □

Lemma 13. *[1, 2] There are $O(rn)$ pairs of sites (x,y) that belong to a common region of an order r Voronoi diagram. The diagram can be found, and all such pairs can be enumerated, in time $O(r^2n + n\log n)$.*

Theorem 3. *The k best spanning trees for a set of points in the plane can be found in time $O(k^2n + n\log n)$.*

Proof: Lemmas 12 and 13 show that in the time bound we can reduce the problem to a graph problem in a graph with n vertices and $O(kn)$ edges. Then by theorem 1 we can find the k best spanning trees in time $O(kn\log\beta(kn,n)+k^2) = O(kn + n\log n + k^2) = O(k^2n + n\log n)$. □

5 Faster Euclidean Spanning Tree Construction

In theorem 3, we did not make much use of the results of the first two sections, wherein we showed how to reduce the number of edges that need be considered to find the k best spanning trees. Indeed, after the reduction to a graph with $O(kn)$ edges, we could have used the algorithm of [18] instead of theorem 1, and still achieved the same final bound. Therefore it should not come as a great surprise that we can improve this bound for certain ranges of the parameters k and n.

First, recall that the Euclidean MST of the points can be computed in time $O(n\log n)$. If we could compute $r(e)$ for each edge e of the MST, then by lemma 6 we could show that all but k of the MST edges must remain in all the k best spanning trees. This may be done as follows.

Lemma 14. *For each replacement edge $r(e)$ connecting points x and y, it must be the case that x and y are part of a set S forming a region $v(S)$ in the order-3 Voronoi diagram of the input points.*

Proof: Let e divide the MST T into components T_1 and T_2, with x in T_1 and y in T_2. Construct the circle C with diameter xy. If C contains a site z which is neither x, y, nor an endpoint of e, then as in lemma 12 either $T-e+(x,z)$ or $T-e+(z,y)$ is better than $T-e+(x,y)$, contradicting the assumption that $r(e)=(x,y)$. The only other way that C could contain four sites would be if they were x and y together with the two endpoints of e; but then again we would have a better replacement for e than (x,y). Therefore C contains at most three sites, and x and y belong to the Voronoi region containing the center of C. □

Corollary 1. *We can compute $r(e)$ for each edge e in the MST, in time $O(n \log n)$.*

Proof: The order-3 Voronoi diagram can be constructed in this time bound. It contains $O(n)$ pairs (x, y) sharing a Voronoi region; therefore by lemma 4 each value of $r(e)$ can be computed in a total time bound of $O(n\alpha(n)) = O(n \log n)$. □

Now as before we can find a set S of all but k edges in the MST, such that the best k spanning trees each contain all the edges in S. Denote the remaining MST edges by e_1, e_2, ..., e_k. Now we must show how to use this information to reduce the possible non-MST edges we must consider.

If we were to follow the graph algorithm, we would want to compute, for each edge, the best replacement edge, and only choose the k edges minimizing the difference in costs between themselves and their replacements. However we do not have time to consider all edges in what is still almost the complete graph. Instead, we first note that any replacement edges considered must come from the k MST edges chosen above. We will find a set of $O(k)$ vertices such that the useful non-MST edges will have both endpoints in this set. These vertices are found by computing, for each vertex, the minimum difference between the costs of a non-MST edge adjacent to the vertex and that edge's replacement, assuming that replacement is one of the k selected edges. We then show how the problem may be reduced to that of computing *bichromatic nearest neighbors*.

Given a point x, and a MST edge e_i, let e_i divide the MST into components T_1 and T_2 with x in T_1; then we define $f_i(x)$ to be the nearest point to x in T_2. We further define

$$F(x) = \min_i w(x, f_i(x)) - w(e_i). \tag{1}$$

Lemma 15. *For each x, $F(x) + w(T)$, where T is the MST, gives the minimum weight of a spanning tree containing all the edges in S and containing an edge not in S adjacent to x.*

Proof: By lemma 8, the spanning tree defined by the lemma must be of the form $T - e_i + (x, y)$ for some i and y. But this is exactly what is minimized by $F(x)$. □

We now show how to compute $F(x)$; this depends on the following well-known solution to the bichromatic nearest neighbor problem:

Lemma 16. *Given a set of m red points, and a set of n blue points, we can compute for each blue point the nearest red point, in time $O((n + m) \log m)$.*

Proof: We simply construct the Voronoi diagram of the set of red points, and use a planar point location procedure [20]. □

Lemma 17. *All values of $F(x)$ can be computed in time $O(nk \log(n/k))$.*

Proof: Removing the edges e_i divides the MST T into $k + 1$ components T_i. We first compute, for each input point, the nearest point in each component. This takes time $O(n \sum \log |T_i|)$; the sum in this bound is maximized when all T_i are equal in size, and the bound then becomes $O(nk \log(n/k))$.

The components T_i, connected by the edges e_i, can be imagined as forming a tree with $(k + 1)$ vertices; there are $2k$ subtrees that can be formed by removing any edge of the

tree. For each point, we find the nearest point to it in each of those subtrees; this may easily be done in time $O(k)$ per point by dynamic programming. Therefore this step takes time $O(nk)$.

At this point we have computed $f_i(x)$ for each i and x, as the nearest point to x in the subtree not containing x of the two formed by removing edge e_i from the component tree. From this $F(x)$ may be computed directly from formula 1, in time $O(nk)$. □

Theorem 4. *The k best spanning trees for a set of points in the plane can be computed in time $O(k^2 + kn \log(n/k))$.*

Proof: The algorithm of corollary 1 takes time $O(n \log n)$, which is always dominated by the $O(kn \log(n/k))$ time to compute $F(x)$. By lemma 11, the k best spanning trees together contain only k edges not already in the MST, and (counting the MST edges not in S) $2k$ edges not in S. Therefore only the $4k$ points having the lowest values of $F(x)$ may be endpoints of those edges. By using lemma 17 and a linear time selection algorithm [4] we may find those $4k$ points in time $O(kn \log(n/k))$. These points determine $O(k^2)$ possible non-MST edges. Therefore, the problem becomes one of finding the k best spanning trees in a graph with $O(k^2)$ edges and $O(k)$ vertices; this can be solved in time $O(k^2)$. □

6 Alternate Metrics

All the algorithms described above depend only on some simple properties of the Voronoi diagram, and therefore work just as well for alternate metrics in the plane. However the fastest construction of the order r Voronoi diagram that works for general metrics takes time $O(k^2 n \log n)$, and therefore theorem 3 never leads to a better time than the $O(k^2 + kn \log n)$ bound of theorem 4.

However in certain cases we can do better. In particular, for the L_1 (equivalently, L_∞) metric, corresponding to the important case of orthogonal spanning trees, we can achieve a time bound of $O(k^2 + kn \log \log n + n \log n)$; this is always at least as fast as our algorithms for the usual L_2 metric.

The algorithm we use is essentially the same as that of theorem 4. The key difference is in the computation of $f_i(x)$, which was the only step in that theorem requiring time $O(kn \log n)$. Recall that this can be considered to be $O(k)$ computations of the bichromatic nearest neighbor problem. We now give an improved solution to this problem for the L_1 metric, after an $O(n \log n)$ time preprocessing (sorting) stage. This leads to a speedup of our spanning tree algorithm.

Recall that the L_1 distance between points (x, y) and (x', y') is simply $|x - x'| + |y - y'|$. We will solve the following simplified version of the problem:

Lemma 18. *Given a set of m red points, and n blue points, sorted by their values of x (x'), and within the same value of x in order by y, we can compute for each blue point (x, y) the nearest red point (x', y') with $x' \leq x$ and $y' \leq y$, in time $O((n + m) \log \log m)$.*

Proof: We process the points in the sorted order. The preprocessing stage consists of simply sorting the points in this order. At any stage in the processing, corresponding to

some particular value of x, we maintain a data structure listing, for each value of y, the nearest red point (x', y') with $x' \leq x$ and $y' \leq y$. This point must already have been processed by the order of processing. Then to find the nearest neighbor of an input point, we perform a lookup in the data structure; to process a red point, we update the data structure.

Each red point (x', y') nearest to any blue point must correspond to some interval $[y', y]$ in the data structure; this is because if some other red point is nearer to a particular value of (x, y) it will be nearer to all blue points with greater values of y. We may represent this collection of intervals using the *flat tree* integer searching data structure of van Emde Boas [22], indexed by the m possible y coordinates of the points in A. Computing these indices for the n input points can be done by a linear time sweep of the points ordered by their y coordinates.

Then finding the nearest red neighbor of a blue point (x, y) simply consists of looking up y in the data structure to find which interval contains it. Processing a red point (x', y') consists of again finding the interval containing y', then splitting that interval at y' to create a new interval for that point, and while each succeeding interval corresponds to a point (x'', y'') farther from (x, y'') than (x', y'), deleting those succeeding intervals and merging them into the new interval for (x', y').

All these data structure operations can be performed in time $O(\log \log m)$ each. A deletion can be charged to its corresponding insertion, so processing each point requires a constant amortized number of data structure operations. Therefore the whole operation takes time $O(n \log \log m)$. □

This algorithm is essentially identical to one used by Eppstein et al [11] to compute RNA secondary structure predictions; they also used a more complicated version of the algorithm as part of a method of computing optimal sequence alignments. Our algorithm should also be compared with that of Chew and Fortune [9] which computes the orthogonal Voronoi diagram of a set of points in $O(n \log \log n)$ time; our algorithm differs in simultaneously performing point location within the constructed Voronoi diagram.

Theorem 5. *The k best orthogonal spanning trees for a set of n points in the plane can be found in time $O(k^2 + kn \log \log(n/k) + n \log n)$.*

Proof: The algorithm of lemma 18 may be repeated four times, in four different directions, to solve the L_1 bichromatic nearest neighbor problem. This problem in turn is solved k times as in theorem 4. The points need be sorted only four times, for the four different directions; the same sorted orders are used in each of the k bichromatic nearest neighbor problems. All other steps are the same as in lemma 17 and theorem 4, using the bound of lemma 18 instead of that of lemma 16. □

Acknowledgements

This research was performed at the Xerox Palo Alto Research Center. I would like to thank Frances Yao for suggesting the Euclidean version of this problem and directing me to reference [2].

References

[1] A. Aggarwal, L.J. Guibas, J. Saxe, and P.W. Shor, A linear time algorithm for computing the Voronoi diagram of a convex polygon, 19th ACM Symp. Theory of Computing, 39–47, 1987.

[2] A. Aggarwal, H. Imai, N. Katoh, and S. Suri, Finding k Points with Minimum Diameter and Related Problems, J. Algorithms, to appear.

[3] A. Aggarwal and J. Wein, Computational Geometry, MIT LCS Research Seminar Series 3, 1988.

[4] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan, Time Bounds for Selection, J. Comput. Syst. Sci. 7, 1972, 448–461.

[5] H. Booth and J. Westbrook, Linear Algorithms for Analysis of Minimum Spanning and Shortest Path Trees in Planar Graphs, Tech. Rep. TR-763, Department of Computer Science, Yale University, Feb. 1990.

[6] R.N. Burns and C.E. Haff, A Ranking Problem in Graphs, 5th Southeast Conf. Combinatorics, Graph Theory and Computing 19, 1974, 461–470.

[7] P.M. Camerini, L. Fratta, and F. Maffioli, The k Shortest Spanning Trees of a Graph, Int. Rep. 73-10, IEEE-LCE Politechnico di Milano, Italy, 1974.

[8] D. Cheriton and R.E. Tarjan, Finding Minimum Spanning Trees, SIAM J. Comput. 5, 1976, 310–313.

[9] L.P. Chew and S. Fortune, Sorting helps for Voronoi diagrams, 13th Symp. Mathematical Progr., Japan, 1988.

[10] D. Eppstein, Offline Algorithms for Dynamic Minimum Spanning Tree Problems, 2nd Worksh. Algorithms and Data Structures, 1991, to appear.

[11] D. Eppstein, Z. Galil, R. Giancarlo, and G.F. Italiano, Sparse Dynamic Programming, 1st ACM-SIAM Symp. Discrete Algorithms, San Francisco, 1990, 513–522.

[12] D. Eppstein, G.F. Italiano, R. Tamassia, R.E. Tarjan, J. Westbrook, and M. Yung, Maintenance of a Minimum Spanning Forest in a Dynamic Planar Graph, 1st ACM-SIAM Symp. Discrete Algorithms, to appear.

[13] G.N. Frederickson, Data Structures for On-Line Updating of Minimum Spanning Trees, with Applications, SIAM J. Comput. 14(4), 1985, 781–798.

[14] G.N. Frederickson, Ambivalent Data Structures for Dynamic 2-edge-connectivity and k Smallest Spanning Trees, 32nd IEEE Conf. Foundations of Computer Science, 1991, to appear.

[15] H.N. Gabow, Two Algorithms for Generating Weighted Spanning Trees in Order, SIAM J. Comput. 6, 1977, 139–150.

[16] H.N. Gabow, Z. Galil, T.H. Spencer, and R.E. Tarjan, Efficient Algorithms for Minimum Spanning Trees on Directed and Undirected Graphs, Combinatorica 6, 1986, 109–122.

[17] H.N. Gabow and M. Stallman, Efficient Algorithms for Graphic Matroid Intersection and Parity, 12th Int. Conf. Automata, Languages, and Programming, Springer-Verlag LNCS 194, 1985, 210–220.

[18] N. Katoh, T. Ibaraki, and H. Mine, An Algorithm for Finding k Minimum Spanning Trees, SIAM J. Comput. 10, 1981, 247–255.

[19] E.W. Mayr and C.G. Plaxton, On the spanning trees of weighted graphs, manuscript, 1990.

[20] N. Sarnak and R.E. Tarjan, Planar Point Location using Persistent Search Trees, C. ACM 29(7), 1986, 669–679.

[21] R.E. Tarjan, Applications of Path Compression on Balanced Trees, J. ACM 26, 1979, 690–715.

[22] P. van Emde Boas, Preserving Order in a Forest in Less than Logarithmic Time, 16th IEEE Symp. Found. Comput. Sci., 1975, and Info. Proc. Lett. 6, 1977, 80–82.