Offline Algorithms for Dynamic Minimum Spanning Tree Problems

David Eppstein

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

Tech. Report 92-04

January 10, 1992

Abstract

We describe an efficient algorithm for maintaining a minimum spanning tree (MST) in a graph subject to a sequence of edge weight modifications. The sequence of minimum spanning trees is computed offline, after the sequence of modifications is known. The algorithm performs $O(\log n)$ work per modification, where n is the number of vertices in the graph. We use our techniques to solve the offline geometric MST problem for a planar point set subject to insertions and deletions; our algorithm for this problem performs $O(\log^2 n)$ work per modification. No previous dynamic geometric MST algorithm was known.

1 Introduction

For many years, algorithm researchers have studied problems of maintaining information about a dynamically changing graph. A classical problem in this field is maintaining the minimum spanning tree (MST) of a graph in which the weights of individual edges are subject to change [1, 5, 6, 10].

Each such update causes at most one edge to leave the MST and at most one other edge to take its place; one might expect algorithms for computing these changes to be quite efficient. Indeed, the best known algorithm takes $O(\sqrt{m})$ time per update, for a graph with n vertices and m edges [6]. However there is clearly room for improvement in this case. For planar graphs the situation is better; one can compute the changes to the MST in time $O(\log n)$ per update [5, 9].

All of the above algorithms are *online*: They accept as input one update at a time, and must output the corresponding changes to the MST before the next update is available to them. It is natural to consider *offline* algorithms for the same problem; such an algorithm would be given a single input consisting of a long sequence of updates, and only after the entire sequence is known would it output the corresponding sequence of MST changes. Offline algorithms are less general than online ones: an offline algorithm can not maintain the MST when updates depend on previous results of the algorithm. In exchange for this loss of flexibility one might expect improved time bounds. Surprisingly, there seems to be no previous work on offline dynamic MST algorithms.

This paper presents such an algorithm. The time per update is $O(\log n)$, greatly improving the previous $O(\sqrt{m})$ bound for the online problem and even matching the best known time bound for planar graphs.

We also examine the problem of maintaining a MST of a dynamic planar point set. It is well known that the MST is a subgraph of the Delaunay triangulation; therefore the static problem can be solved in time $O(n \log n)$. However no efficient algorithm was known for the dynamic geometric MST problem, in which updates consist of insertions and deletions of single points. We use techniques similar to those in our graph algorithm, and the graph algorithm itself as a subroutine, to achieve $O(\log^2 n)$ update time for the offline dynamic geometric MST problem. This time bound can be improved to $O(\log n \log \log n)$ for rectilinear MSTs.

2 Reduction and Contraction

Our offline graph MST algorithm works in a series of phases. In each phase we perform two steps: *reduction* and *contraction.* In a given phase we divide the sequence of edge weight updates into *blocks* and perform the reduction and contraction operations separately for each block.

Reduction consists of finding a set of edges in the graph which, given the update operations in the block, can not be used in any of the MSTs of any of the sequence of weighted graphs corresponding to the updates. Once these edges are found, they can be removed from the graph without changing the results of the computation in that block. Thus the graph can be made to have a smaller number of edges.

Contraction, similarly, consists of finding a set of edges which must be used in all the MSTs for the block. We then *contract* each of these edges, by merging sets of vertices that are connected by those edges. Thus the graph can be made to have a smaller number of vertices.

Reduction and contraction were used in a previous paper in which we described algorithms for finding a set of several different spanning trees having the minimum possible total weight [2]. However in that paper the actual process of reduction and contraction is performed differently. Curiously enough the problem solved in that paper had previously been attacked by using the (online) dynamic MST problem as a subroutine [6].

Both reduction and contraction are implemented in this paper using (static) MST computations. For this purpose we will use the recently discovered linear time MST algorithm of Fredman and Willard [7]. This algorithm uses a nonstandard model of computation, in which edge weights are binary integers; other MST algorithms do not specify the representation of weights, and only operate on them by comparisons. In our algorithm we allow $O(\log n)$ time per update, and we never operate on more than $O(n^2)$ updates at once. Therefore we have time to sort the weights using any $O(n \log n)$ time comparison based algorithm, and convert them to the integer representation needed by the MST algorithm.

Edge insertion and deletion operations can also be handled, by treating a deleted edge as having infinite cost. The obvious implementation of this would cause the algorithm to use $\Omega(n^2)$ space to keep track of the costs of all edges; we will show how to avoid this space penalty.

3 Reduction

The following fact is the basis of the reduction step.

Lemma 1. *Let G be a weighted graph, and S be a subset of its edges. Let T be the MST of $G - S$. Then no matter how the edge weights in S are changed, the MST of G will only contain edges in $T \cup S$.*

Proof: This follows immediately from the standard "dual greedy" MST algorithm in which the highest weight edge in any cycle is removed until the remaining graph has no cycles. If we break the cycles in $G - S$ first, all edges in $G - S - T$ will be removed. Therefore all edges in the MST will be in the remaining graph, $T \cup S$. $\square$

Lemma 2. *Assume we are performing a reduction step in a block of k updates. Then we can reduce the number of edges in the graph from m to at most $n + k - 1$, in time $O(m)$.*

Proof: Let S be the set of edges updated in the block. Compute the MST of $G - S$ and apply Lemma 1. Then we need only keep the $n - 1$ MST edges, together with the edges in S; Lemma 1 shows that throwing away the other edges will not change the results of the computation in this block. $\square$

4 Contraction

The following fact is the basis of the contraction step.

Lemma 3. *Let G be a weighted graph, and S be a subset of its edges. Let T be the MST of G, when the edges of S are given weights lower than those of any other edge in the graph. Then no matter how the edge weights in S are changed, the MST of G will always contain the edges in $T - S$.*

Proof: Consider changing the weights of the edges in S, one by one, from the weights in T to the new desired weight. At each such change, either the MST will not change, or the changed edge will leave the MST and some other edge will replace it. Therefore, the edges in $T - S$ will remain in the MST. $\square$

Lemma 4. *Assume we are performing a contraction step in a block of k updates. Then we can reduce the number of vertices in the graph from n to at most $k+1$, in time $O(m)$.*

Proof: Let S be the set of edges updated in the block. We assume that Lemma 2 has already been applied, so $G-S$ is a tree. Weight the edges in S lower than the minimum weight in $G-S$ and compute the MST T of G.

Construct a new graph G' as follows. Create a new vertex for each connected component of $T-S$, and replace each edge (x,y) in the graph with (x',y') where x' and y' are the vertices corresponding to the components containing x and y respectively.

Lemma 3 shows that each MST of G, for a given assignment of weights to the edges of S, can be found by computing the MST of G', and taking the union of the corresponding edges in G with the edges in $T-S$. In particular, each individual change to the MST of G (consisting of the removal of one edge from the MST, and the addition of one replacement edge) corresponds exactly to such a change in the MST of G', and vice versa. $\square$

5 The Graph Algorithm

We now solve the offline dynamic graph MST problem. Recall that the algorithm is divided into *phases*, in each of which we perform reduce and contract steps within *blocks* of edge weight update operations. It remains to specify how many phases to use, and what block size to use within a phase.

We begin with all blocks of size m, the largest number of edges in the graph at any one time. Contraction and reduction are no help for such a large block, so we do not perform these steps in the first phase. Within a block we treat insertions and deletions as changes involving infinite edge weights, as discussed previously; the initial selection of block size m instead of $n^2/2$ means we only need $O(m)$ space instead of $O(n^2)$. At this point we can sort the weights used within the block (including the edge weights on entry to the block) in preparation for the MST algorithm of Fredman and Willard [7].

In each succeeding phase, we start with blocks of size b, and in each block the graph has been reduced, contracted, and reduced again. Therefore by Lemmas 2 and 4 it has at most $b+1$ vertices and b edges not involved in updates. There are of course at most b edges involved in updates. We then split each block into two smaller blocks of approximately equal length. These blocks will be used in the new phase, in which we again reduce, contract, and reduce the graph.

After $O(\log n)$ such phases, each block will consist of a single update and the reduced graph in each block will have two vertices and one non-updated edge, at which point we can easily tell whether the updated edge replaces or is replaced by the other edge.

This gives us a sequence of edge replacements, which we translate back into the unreduced form to produce the sequence of MST changes corresponding to the initial sequence of edge cost updates in the original graph.

Theorem 1. *Given a sequence of k edge weight modifications in a graph, starting from a state in which all weights are equal, we can compute the corresponding sequence of minimum spanning trees in time $O(k \log n)$ and space $O(m)$.*

Proof: The correctness of the algorithm follows from Lemmas 2 and 4. In each phase, for each block of size b, we perform $O(b)$ operations; therefore, for each update we perform $O(1)$ operations. There are $O(\log n)$ phases, so for each update we perform $O(\log n)$ total work. $\Box$

6 The Geometric Algorithm

We now describe how to solve the offline MST problem for a planar point set, with updates consisting of point insertions and deletions. The geometric MST is simply the MST of the complete graph on the points, with edges weighted by distance. However each point update corresponds to the insertion or deletion of $O(n)$ edges in the complete graph, so a direct application of our graph algorithm would be no more efficient than recomputing the MST using the $O(n \log n)$ time static algorithm after each update.

Instead we use the following approach. We recursively break our update sequence into blocks, as in the graph algorithm. The processing at each level consists of identifying certain pairs of points as "interesting", adding the corresponding edges to a graph problem, and removing some of the points from lower levels of the recursion. After this decomposition, we will have identified $O(n \log n)$ interesting edges in each block of length n; all MST edges will occur in this list of interesting edges. Each such edge can be considered to be inserted in the graph when both of its two endpoints have been inserted in the plane, and deleted when one of its endpoints is deleted.

Thus we reduce the problem to a graph problem in which there are $O(\log n)$ edge updates per point insertion or deletion, and which can therefore be solved in time $O(\log^2 n)$ per point update.

We assume throughout this section that point distances are measured using the Euclidean L_2 metric; similar techniques apply to other commonly used metrics. Define a *sextant* from point x to be an infinite closed wedge with $60°$ angle having x as its corner, with either one side of the wedge horizontal or both sides at $60°$ angles from horizontal. Each point corresponds to six possible sextants, and these six sextants together cover the plane. Our identification of interesting edges is based on the following well known facts.

Lemma 5. *Let (x, y) be an edge in the geometric MST of a planar point set S. Then y is the nearest point to x in the sextant from x containing y.*

Proof: Let z be closer. Then the edges xz and zy are both shorter than xy, so xy could not be a MST edge. □

Lemma 6. *Given sets S and T, the nearest points in S in each sextant from each point of T can be found in time $O((|S| + |T|) \log |S|)$.*

Proof: For each sextant direction, one can construct in time $O(|S| \log |S|)$ an appropriate Voronoi diagram of the points of S. Point location queries in such a diagram can be performed for each point of T, in time $O(\log |S|)$ per query. Six such diagrams need be computed, for the six possible sextant directions, and six such queries need be performed for each point of T. □

We now describe our offline geometric MST algorithm. As before, we consider blocks of update operations, perform certain reduction steps on them, and recursively divide them into pairs of smaller blocks.

Our block reduction works as follows. Let S be the *static* points of the block; that is, those points that are not inserted or deleted by the updates in the block. Let T be the *dynamic* points, that are updated within the block. Using lemma 6 we compute, for each point in $S \cup T$, the nearest points in S in each sextant. We declare each pair of points found in this step to be an interesting edge. Finally, we remove S from the point set when we consider the two recursive subblocks created by splitting our block. Thus in the recursive processing, the set of static points in each subblock will be a subset of the dynamic points in the other subblock.

Theorem 2. *Given a sequence of k point insertions and deletions in the plane, starting from an empty plane, we can compute the corresponding sequence of minimum spanning trees in time $O(k \log^2 n)$ and space $O(n)$, where n is the maximum number of points in the plane at any one time.*

Proof: We start with blocks of size $O(n/\log n)$. In the first reduction, there are $O(n)$ static points and therefore the time taken is $O(n \log n)$ per block. In subsequent reductions, when the block size is m, the number of static points is at most $m+1$, the maximum size of the other block into which the block's parent is split; therefore, the time taken is $O(m \log m)$. The total work within each initial block per level of the recursive decomposition is $O(n)$, and the total work in all the $O(\log n)$ levels is $O(n \log n)$. Once the decomposition is performed, we will have identified at most six interesting edges per point per level, or $O(n)$ interesting edges altogether. Solving the offline graph MST problem on the sequence of insertions and deletions of interesting edges takes time $O(n \log n)$. Therefore the total time per block is $O(n \log n)$, and the time per update is $O(\log^2 n)$.

It remains to show that all MST edges are included in the set of interesting edges, and therefore that the algorithm correctly computes the sequence of MSTs for the changing point set. Let (x, y) be an edge in the MST after change c, and consider the largest block b containing c in which at least one of x and y would be static (if it weren't thrown away in a larger block). Such a block must exist, because only one point can be dynamic in the block containing c alone. Then neither x nor y can have been thrown away in a larger block, because only static points are thrown away. Assume without loss of generality that x is static in b. By lemma 5, x is the closest point to y in its sextant at time c. Since the static points of b are a subset of the input points existing after change c, point x must be the static point in block b that is closest to y in its sextant. Therefore it is included in the set of interesting edges. $\square$

7 Rectilinear Spanning Trees

We can improve theorem 2 for the important case of *rectilinear* minimum spanning trees. These are MSTs for the L_1 (or equivalently L_∞) planar metric. The following analogues of lemmas 5 and 6 hold; lemma 8 below is proved in [2]. We define a *quadrant* to be a quarterplane bounded by one horizontal and one vertical line.

Lemma 7. *Let (x, y) be an edge in the rectlinear MST of a planar point set S. Then y is the L_1-nearest point to x in the quadrant from x containing y.* $\square$

Lemma 8. *Given sets S and T, in two sorted orders, one for each coordinate, the L_1-nearest points in S in each quadrant from each point of T can be found in time $O((|S| + |T|) \log \log |S|)$.* $\square$

Lemma 8 speeds up the time spent identifying interesting edges. However there is another bottleneck in our algorithm, which is the $O(\log n)$ time spent processing each interesting edge. We deal with this by interleaving the graph edge reduction of theorem 2 with our geometric point reduction. To do this we maintain a *mixed* problem, consisting of a graph in which some of the vertices are geometric points, some of the vertices are just vertices, pairs of geometric points can be connected by their L_1-distances, and all vertices can be connected by the edges in the graph.

Lemma 9. *Given a mixed problem consisting of $O(n)$ points, non-point vertices, and edges, with the points sorted by each coordinate, the mixed rectilinear MST can be computed in time $O(n \log \log n)$.*

Proof: We compute a set of $O(n)$ interesting point-point distances using lemma 8, in time $O(n \log \log n)$. This gives us a pure graph problem with $O(n)$ vertices and edges, which can be solved using any $O(n \log \log n)$-time MST algorithm. $\square$

We cannot use the linear time MST algorithm of Fredman and Willard [7] because the coordinates of the input points are not assumed to be integers. However the best known non-integer MST algorithm takes time $O(n \log \log^* n)$ for our problem [8], easily meeting the $O(n \log \log n)$ requirement.

Theorem 3. *Given a sequence of k point insertions and deletions in the plane, starting from an empty plane, we can compute the corresponding sequence of rectilinear minimum spanning trees in time $O(k \log n \log \log n)$ and space $O(n)$, where n is the maximum number of points in the plane at any one time.*

Proof: We begin with blocks of size n. Within each initial block, we sort the points by each coordinate; these sorted orders will be maintained in the smaller blocks recursively split from the initial blocks. In each block, we maintain a mixed problem such that the following conditions hold.

- The MST of the mixed problem is a contraction of the MST of all static points in the block.
- Each edge in the MST of all static points that is removed by some change in the block corresponds either to a pair of geometric points in the mixed problem, or to an edge in the mixed problem.
- For each edge in an MST in the block that connects a static and dynamic point, the static point is one of the geometric points in the mixed problem.
- The size of the mixed problem is proportional to the size of the block.

In blocks consisting of a single change, we have a mixed problem of constant size in which one geometric point is inserted or deleted; the appropriate MST update can be found in constant time.

In larger blocks, we split each block into two smaller blocks and reduce the mixed problem appropriately. This is done as in the graph algorithm by computing two MSTs, one of the mixed problem without any dynamic points, and one of the mixed problem with all dynamic points. We also compute all interesting edges consisting of the closest static point in each quadrant to each dynamic point.

Then as in the graph algorithm we can remove all graph edges not in the first MST, and contract the edges that are in both MSTs. Whenever a contraction involves a geometric point, that point must become a graph vertex instead; therefore we cannot contract all edges that are in both MSTs. Instead, we mark all static points that are the nearest in some quadrant to some dynamic point; edges touching marked points will not be contracted. Because of lemma 7, in a block of b changes at most $4b$ static points will be marked, and each marked point can protect at most 4 edges from contraction. At most $3b$ static MST edges can be replaced when we include the dynamic points. Therefore the contraction results in a mixed problem having at most $19b$ edges and vertices, so the problem size is reduced appropriately. It can be verified that all the conditions above are maintained. The reduction process takes time $O(b \log\log b)$, for a total time of $O(\log n \log\log n)$ per change. $\square$

8 Conclusion

We described efficient algorithms for offline computation of MSTs in a changing graph or point set. We use reduction and contraction, which we introduced in our paper on finding several small spanning trees [2], and have recently applied to a persistent query version of the dynamic MST problem [3]. Reduction and contraction have thus proven useful for a variety of MST problems; perhaps they can be used in other graph algorithms.

The most pressing open problem suggested by this work is maintenance of geometric MSTs. No algorithm was known that achieved sublinear time bounds for both point insertions and deletions. We achieve $O(\log^2 n)$ time, at the expense of requiring offline operation. In an earlier version of this paper, we noted that our results may lead the way to efficient online algorithms for this problem. Subsequent to this research, we have found such an algorithm: we can maintain rectilinear MSTs in time $O(\sqrt{n}\log^3 n)$ per update, and Euclidean MSTs in time $O(n^{5/6}\log^{1/2} n)$ per update [4].

References

[1] F. Chin and D. Houck. Algorithms for updating minimum spanning trees. *J. Comput. Syst. Sci.* 16 (1978) 333–344.

[2] D. Eppstein. Finding the k smallest spanning trees. *Proc. 2nd Scand. Worksh. Algorithm Theory*, Springer-Verlag LNCS 447 (1990) 38–47; *BIT*, to appear.

[3] D. Eppstein. Persistence, offline algorithms, and space compaction. Tech. Rep. 91-54, Dept. of Information and Computer Science, Univ. of California, Irvine, CA 92717.

[4] D. Eppstein. Fully dynamic maintenance of Euclidean spanning trees. Manuscript.

[5] 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. *Proc. 1st ACM/SIAM Symp. Discrete Algorithms* (1990) 1–11; *Algorithmica*, to appear.

[6] G.N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.* 14 (1985) 781–798.

[7] M.L. Fredman and D.E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Proc. 31st IEEE Symp. Found. Computer Science* (1990) 719–725.

[8] H.N. Gabow, Z. Galil, T. Spencer, and R. Tarjan. Efficient algorithsms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica* 6 (1986) 109–122.

[9] H.N. Gabow and M. Stallman. Efficient algorithms for graphic matroid intersection and parity. *Proc. 12th Int. Conf. Automata, Languages, and Programming*, Springer-Verlag LNCS 194 (1985) 210–220.

[10] P.M. Spira and A. Pan. On finding and updating spanning trees and shortest paths. *SIAM J. Comput.* 4 (1975) 375–380.