Dynamic Euclidean Minimum Spanning Trees and Extrema of Binary Functions

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

Abstract

We maintain the minimum spanning tree of a point set in the plane, subject to point insertions and deletions, in amortized time $O(n^{1/2}\log^2 n)$ per update operation. We reduce the problem to maintaining bichromatic closest pairs, which we solve in time $O(n^{\epsilon})$ per update. Our algorithm uses a novel construction, the *ordered nearest neighbor path* of a set of points. Our results generalize to higher dimensions, and to fully dynamic algorithms for maintaining minima of binary functions, including the diameter of a point set and the bichromatic farthest pair.

1 Introduction

A dynamic geometric data structure is one that maintains the solution to some problem, defined on a geometric input such as a point set, as the input undergoes update operations such as insertions or deletions of single points. Dynamic algorithms have been studied for many geometric optimization problems, including closest pairs [7, 23, 25, 26], diameter [7, 26], width [4], convex hulls [15, 22], linear programming [2, 9, 18], smallest k-gons [6, 11], and minimum spanning trees (MSTs) [8]. Many of these algorithms suffer under a restriction that, if deletions are allowed at all, they may only occur at certain prespecified times—the algorithms are not *fully dynamic*.

A number of other papers have considered dynamic computational geometry problems under an average case model that assumes that among a given set of points each point is equally likely to be inserted or deleted next [10, 19, 20, 21, 24]. However we are interested here in worst case bounds.

We are particularly interested in the dynamic geometric MST problem. If only insertions are allowed, it is not hard to maintain the MST in time $O(\log^2 n)$ per update. The same bound has recently been achieved for *offline* updates consisting of a prespecified sequence of insertions and deletions [8]; this can be further improved for rectilinear metrics. No known online algorithm allowing both insertions and deletions (or even deletions only) had significantly better worst case bounds than the trivial $O(n \log n)$ method of recomputing the MST from scratch after each update.

Following Agarwal *et al.* [1], we reduce MST maintenance to the *bichromatic closest pair* problem. Suppose we are given two point sets: red points R and blue points B. The bichromatic closest pair is the pair of two points $p \in R$ and $q \in B$, minimizing the distance pq. In other words, $d(p, q) = d(R, B)$. The dynamic geometric MST problem can be reduced to dynamically maintaining the solutions to several bichromatic closest pair problems, and to solving a dynamic graph MST problem in which edges are inserted or deleted according to the behavior of the closest pairs. If we were performing insertions only, maintaining bichromatic closest pairs would not be difficult. Vaidya [27] described a method for performing insertions in the red set, and both insertions and deletions in the blue set, in time $O(n^{1/2} \log n)$ per operation. But no fully dynamic algorithm for this problem was previously known.

A generalization of bichromatic closest pairs was described by Dobkin and Suri [7], who describe methods for computing the minimum of any binary function $f(x, y)$, with x and y ranging over all values in input sets X and Y. Dobkin and Suri gave algorithms for maintaining such minima in a semi-online model of computation, in which as each point is inserted we are told the time it will be deleted;

their algorithm needs only the existence of a data structure that, given a value y, finds the value x among the given inputs that minimize $f(x, y)$. This class of functions includes the closest pair problem, the diameter problem mentioned earlier, and many similar geometric optimization problems. Dobkin and Suri call this problem that of finding minima of decomposable functions, however this terminology is unfortunate as the word "decomposable" has been used in the literature to mean something different [5]: a function f on sets is decomposable if whenever S is the disjoint union of S_1 and S_2, $f(S)$ can be computed easily from $f(S_1)$ and $f(S_2)$. We prefer to call this problem that of finding minima of binary functions. In any case no fully dynamic algorithm for such problems was previously known.

In this paper we provide the first known fully dynamic algorithms for the following geometric optimization problems.

- For any planar point set undergoing point insertions and deletions, and for any $\epsilon > 0$, the Euclidean bichromatic closest pair, bichromatic farthest pair, or diameter can be maintained in amortized time $O(n^{\epsilon})$ per insertion or deletion.
- For any planar point set undergoing point insertions and deletions, the Euclidean MST can be maintained in amortized time $O(n^{1/2} \log^2 n)$ per insertion or deletion.
- For d-dimensional point sets in the rectilinear (L_1 or L_∞) metrics, and for the Euclidean metric in dimension $d \leq 4$, the MST can be maintained in amortized time $O(n^{1/2} \log^d n)$ per update.
- For a point set of any fixed dimension, the Euclidean MST, bichromatic closest pair, bichromatic farthest pair, diameter, minimum or maximum distance between a point and a hyperplane, and minimum or maximum separation between axis-aligned boxes, can all be maintained in amortized time $O(n^{1-\epsilon})$ per update, where $\epsilon > 0$ is some constant depending on d.

More generally, we maintain the minimum or maximum of any binary function, given a data structure that answers nearest neighbor queries for that function and that allows fully dynamic point insertions and deletions. For example, our methods solve the problems of the minimum separation among a set of rectangles or higher-dimensional axis-aligned boxes, the minimum or maximum distance between a set of points and a set of hyperplanes, and the minimum or maximum axis-aligned rectangles or boxes having a long diagonal defined by a pair of points.

2 The Dynamic Post Office Problem

We use a data structure for the *post office problem* as a key subroutine in our algorithm. In this problem, one is given a point set S, and a query point p; the problem is to find the nearest point to p in S. If S is unchanging, this can be done by computing a Voronoi diagram of S (in time $O(n \log n)$ for the Euclidean planar metric), and performing point location queries in the diagram ($O(\log n)$ time in the plane).

In our use of this problem, S will undergo point insertions and deletions. Until recently the best known algorithm for such a dynamic post office problem involved partitioning the points into groups, and recomputing the Voronoi diagram in a single group after each update; this approach results in update and query times of $O(n^{1/2} \log^{1/2} n)$. In a recent breakthrough, Agarwal and Matoušek [2, 3] showed how to solve the post office problem by applying parametric search techniques together with a halfspace range searching data structure. They were able to achieve the following result.

Lemma 1 (Agarwal and Matoušek [2, 3]). *For any fixed $\epsilon > 0$, there is a data structure with which we can insert and delete points in a planar set S, and find the nearest neighbors to given query points, in time $O(n^{\epsilon})$ per insertion, deletion, or query.*

Agarwal and Matoušek's technique can also be extended to higher dimensions. For any dimension d they present data structures taking time $O(n^{1-2/(\lceil d/2 \rceil + 1)+\epsilon})$ per operation. Their data structure takes space $O(n^{1+\epsilon})$ for $d = 2$, and has higher space bounds in higher dimensions.

We note that for rectilinear (L_1 and L_∞) metrics, orthogonal range query data structures let us solve the post office problem in time $O(\log^d n)$ per operation [16, 17, 28].

3 Ordered Nearest Neighbors

Suppose we are given a bichromatic set S of red and blue points. We define a *bichromatic ordered nearest neighbor path* to be a path produced by the following sequence of operations. We first choose p_1 arbitrarily. Then we successively extend this path by one more point, by choosing p_{i+1} to be the nearest neighbor to p_i among the unchosen points of the opposite color. If there are m red points and $n > m$ blue points, and we begin the path with a red point, the path will contain $2m$ vertices, after which we are left with $n - m$ unmatched blue points.

Lemma 2. *Let S be a bichromatic point set, with m red points and $n > m$ blue points. Suppose, for distance function $d(p, q)$, the post office problem for S can be solved (after a preprocessing stage taking time $P(n)$) in time $T(n)$ per insertion, deletion, or query. Then we can compute a bichromatic ordered nearest neighbor path for S in time $O(P(n) + mT(n))$.*

Proof: We maintain two post office problem data structures for looking up nearest neighbors among the unchosen points, one data structure for each color. When we choose a point we remove it from the structure. There are at most $2m$ queries and a similar number of deletions, each of which takes time $T(n)$. Therefore the total time is $O(P(n) + mT(n))$. □

The $O(P(n))$ term in the time bound of Lemma 2 is a one time start up cost for building the post office data structure. Once we have computed the nearest neighbor path, we can add back the deleted points in time $O(mT(n))$, matching the time for constructing the nearest neighbor graph. Then we can remember the data structure and re-use it in later computations, avoiding the start-up cost. If the set of n points changes, the post office data structure can be updated in time $O(T(n))$ per change.

4 Maintaining the Bichromatic Closest Pair

In order to maintain the bichromatic closest pair of points, we partition the point set S into *levels* numbered from 0 to $\lceil \log_2 n \rceil$. The points at each level may be of either color. Let S_i be the set of points at level i; S_i will contain at most 2^i points. Initially $S_{\lceil \log n \rceil}$ will contain all of S. For each S_i we maintain a graph G_i. G_i may have as its vertices not only the points in S_i but also some other points in $S - S_i$.

As we will show below, one of these graphs will contain an edge (p, q) such that $d(p, q) = d(R, B)$. Thus we can maintain the closest pair using a priority queue of all the edges in these graphs. We store the edges of each G_i in a separate priority queue Q_i, and determine the overall minimum length edge by examining the $O(\log n)$ minima from the different queues.

G_i is initially constructed as follows. Let $R_i = S_i \cap R$ and $B_i = S_i \cap B$. G_i consists of two ordered nearest neighbor paths, one for $R_i \cup B$ and one for $B_i \cup R$. We keep a post office data structure for all of S; therefore G_i can be constructed in time $O(|S_i|T(n))$ by Lemma 2 and the discussion following it. As the algorithm progresses we delete edges from G_i and periodically reconstruct it from scratch.

We will also periodically reconstruct the overall data structure. If there were m points in S when it was last reconstructed, we reconstruct it after performing $m/2$ update operations. This ensures that, n, the number of points in S is always between

$m/2$ and $3m/2$. The amortized time spent in a global reconstruction is $O(T(n))$ per update, which can be charged to each update operation without affecting the asymptotic running time.

4.1 Inserting and deleting points

Whenever we insert a point p into S, we place p in level 0. Then, as long as p is in a level i containing more than 2^i points, we move all points of level i to level $i+1$, making level i empty. Once p enters a level i in which there are at most 2^i points, we remove all graphs G_j for $j < i$, and reconstruct graph G_i as described above. We also discard all priority queues Q_j, for $j < i$, and reconstruct Q_i storing the edges of new G_i.

We delete a point q from S as follows. We delete all the edges incident to q from each G_i. If we deleted a directed edge of the form (p, q), then we also delete p from its present level, and add p to level 0 as if it were newly inserted. However we do not delete any other edges incident to p. If $q \in S_j$, then there are at most two edges pointing towards q in G_j, and there is at most one such edge in any other graph G_i, $i \neq j$. At most four edges are deleted from level j and at most two from each other level. $O(\log n)$ points are moved to level 0. As in the insertion procedure, we then move these points as a group through successive levels until the level they are in is large enough to hold them, and then reconstruct the graph for the level they end up in.

4.2 Correctness

In order to prove the correctness of the algorithm, we need the following obvious fact.

Lemma 3. *Let i be some level. Then the level of all points, which are inserted to S or moved to level 0 after the most recent construction of G_i, is less than i.*

The correctness of the algorithm now follows from the following lemma.

Lemma 4. *There is an edge (p, q) in one of the graphs G_i such that $d(p, q) = d(R, B)$.*

Proof: Let (p, q) be a bichromatic closest pair in S. Suppose without loss of generality that $p \in R_i$ and $q \in B_j$, and that $j \geq i$. It follows from Lemma 3 that q was in S and p was in S_i when G_i was constructed the last time.

First assume that $q < p$ in the ordering of the nearest neighbor path computed for $R_i \cup B$. Let (q, r) be the outgoing edge from q when that graph was constructed

the last time. By definition of the bichromatic ordered nearest neighbor path, $d(q,r) \leq d(q,\, p)$. If the edge (q,r) still exists, then it is a bichromatic closest pair and the lemma is obviously true, so assume that (q,r) has been deleted. Since q is still in S, the only way (q,r) could be deleted from G_i was that r was deleted from S. In that case, q would have been moved to level 0, which by Lemma 3 implies that $j < i$, a contradiction.

Similarly, assume that $p < q$ in the ordering for $R_i \cup B$, and let (p,r) be the outgoing edge from p in that graph when it was last reconstructed. Then $d(p,r) \leq d(p,q)$, so if (p,r) is still in G_i it is a bichromatic closest pair. If it is not in G_i then r must have been deleted, and p would have been moved to level 0, which by Lemma 3 contradicts the assumption that p is still in level i. □

4.3 Time analysis

We first analyze the time spent reconstructing the entire data structure. Recall that, between periodic reconstructions, there is some value m so that n remains between $m/2$ and $3m/2$. For convenience define N to be $3m/2$. Let $T(n)$ be the time per operation in a nearest neighbor query data structure. We will assume that $T(n)$ is monotonic, and satisfies the property $T(N) = O(T(n))$ (as will be true for any polynomial or other well behaved time bound). The time to perform each periodic reconstruction is then $O(nT(n))$, and each reconstruction happens after $\Omega(n)$ updates, so the amortized time per update is $O(T(n))$.

We define for later use a potential function of a point $p \in S_i$ to be

$$\Phi(p) = cT(N)(\log_2 N + 1 - i)\,,$$

where c is some appropriate constant. Since $i \leq \lceil \log_2 n \rceil$, $\Phi(p)$ is always non-negative. We define the overall potential function $\Phi(S) = \sum_{p \in S} \Phi(p)$. $\Phi(S) = O(nT(n)\log n)$ so the increase in Φ occurring at reconstructions of the overall data structure, amortized per update, is $O(T(n)\log n)$.

Next, let us analyze the time spent in reconstructing G_i. The actual time spent in constructing G_i is $O(2^i T(n))$. We also spend an additional $O(2^i)$ time to reconstruct the priority queue Q_i. But observe that G_i is constructed only when the points from S_{i-1} are moved to S_i. Since the points are moved from S_{i-1} to S_i only if $|S_{i-1}| > 2^{i-1}$, $\Phi(S)$ decreases by at least $c2^{i-1}T(N)$. The amortized time in reconstructing G_i and updating the structure is thus zero if c is chosen sufficiently large.

When we insert a point, we add it to S_0, which increases $\Phi(S)$ by $cT(N)(\log_2 N + 1)$. Since the actual time spent in adding a point is $O(\log n)$ plus the time spent

in reconstructing the appropriate graphs, the amortized running time of an insert operation is $O(T(n)\log n)$.

Deleting a point involves removing at most four edges from each G_i and Q_i, moving $O(\log n)$ points to S_0, and reconstructing appropriate graphs. The total time spent in deleting the edges from G_i and Q_i is $O(\log^2 n)$, and moving the points to S_0 increases the total potential $\Phi(S)$ by $O(\log n)\cdot cT(N)(\log_2 N+1) = O(T(n)\log^2 n)$. Since the amortized time spent in reconstructing the graphs is zero, the total amortized time spent in deleting a point is $O(T(n)\log^2 n)$.

Theorem 1. *Let $d(p,q)$ be a distance function for which some data structure allows us to perform nearest neighbor queries, and insert and delete points, in time $T(n)$ per operation, and let $T(n)$ be monotonic and satisfy $T(3n) = O(T(n))$. Then we can maintain the bichromatic closest distance $d(R,B)$, as well as a pair of points realizing that distance, in amortized time $O(T(n)\log n)$ per insertion, and $O(T(n)\log^2 n)$ per deletion.*

The assumption that $T(n)$ be linear or sublinear is a reasonable one, as in linear time we could answer nearest neighbor queries simply by computing $d(p,q)$ for all input points p. The space requirement for the data structure is $O(n)$ for the paths at each level, plus the amount of space required for the nearest neighbor data structure on a set of N points.

5 Diameter and other Binary Functions

The algorithm described above will work to compute the minimum value of any binary function, not necessarily a metric, as long as we can find an appropriate bichromatic ordered nearest neighbor path, which can be done using a solution to the post office problem. By negating the distance function, we can compute the maximum value in a similar fashion.

The general problem of finding minima and maxima of binary functions is described by Dobkin and Suri [7], who give algorithms for maintaining such extrema in a semi-online model of computation, in which as each point is inserted we are told the time it will be deleted. However no fully dynamic algorithm for these problems was previously known.

The technique as we described it works only for *bichromatic* problems. To extend it to uncolored problems, we replace a single uncolored point set by a colored set containing both one red point and one blue point in place of each uncolored point. However the minimum of a binary function on such sets may be $f(x,x)$ for some x, which we may wish to disallow; e.g. for the uncolored closest pair problem we

wish the points of the pair to be distinct. To avoid this difficulty, we can modify the distance function so that the distance between a point and itself is defined to be $+\infty$. We can use a data structure for the original post office problem to answer queries for the modified distance function as follows: simply remove the query point from the data structure, perform the query, and put any removed point back where it was. With this modification, uncolored closest pairs can be found by letting $R = B = S$. Of course fully dynamic techniques were known for certain specific uncolored closest pair problems [25] but these techniques did not generalize to binary functions or even arbitrary metrics.

We illustrate our technique with some examples of specific bichromatic and uncolored problems of minimizing binary functions.

The *bichromatic closest pair* of a planar point set can be maintained in time $O(n^\epsilon)$ per insertion or deletion, by using the post office data structure cited in Lemma 1. In any higher dimension d, the bichromatic closest pair can be maintained in time $O(n^{1-2/(\lceil d/2\rceil+1)+\epsilon})$ per update.

The *bichromatic farthest pair* of a two-colored planar point set is the farthest distance between any points of opposite colors, and is thus the maximum of the distance function. Using Agarwal and Matoušek's methods [2, 3], a post office data structure for the negation of the Euclidean distance can be maintained in time $O(n^{1-2/(\lceil d/2\rceil+1)+\epsilon})$ per operation. Thus we can use our technique to maintain the bichromatic farthest pair in time $O(n^{1-2/(\lceil d/2\rceil+1)+\epsilon})$ per update.

The *diameter* of a point set is the uncolored version of the farthest pair problem. For this problem we do not even need to modify the distance function as described for uncolored problems in general: if we simply set $R = B = S$ the uncolored and bichromatic farthest pairs will coincide. Thus the diameter can be maintained in the same time bound of $O(n^{1-2/(\lceil d/2\rceil+1)+\epsilon})$ per update.

The same technique applies to the problems of the minimum separation among a set of rectangles or higher-dimensional axis-aligned boxes, the minimum or maximum distance between a set of points and a set of hyperplanes, and the minimum or maximum axis-aligned rectangles or boxes having a long diagonal defined by a pair of points. For each of these problems, Dobkin and Suri [7] give data structures that answer post office queries in sublinear time after polynomial preprocessing time. A static data structure of this type can be transformed to a fully dynamic structure using standard grouping methods, after which we can use our technique to achieve fully dynamic algorithms for maintaining these quantities with sublinear update times.

6 Dynamic Euclidean Minimum Spanning Trees

We have seen how to use the nearest neighbor searching problem to maintain the bichromatic closest pair of a point set, as points are inserted and deleted. We now apply these results in an algorithm for maintaining the Euclidean minimum spanning tree of a point set. The connection between bichromatic closest pairs and minimum spanning trees can be seen from the following lemma.

Lemma 5 (Agarwal *et al.* [1]). *Given a set of n points in R^d, we can form a hierarchical collection of $O(n \log^{d-1} n)$ bichromatic closest pair problems, so that each point is involved in $O(i^{d-1})$ problems of size $O(n/2^i)$ ($1 \leq i \leq \log n$) and so that each MST edge is the solution to one of the closest pair problems.*

Proof sketch: For simplicity of exposition we demonstrate the result in the case that $d = 2$; the higher dimensional versions follow analogously.

If pq is an MST edge, and w is a double wedge having sufficiently small interior angle, with p in one half of w and q in the other, then pq must have the minimum distance over all such pairs defined by the points in w. Therefore if F is a family of double wedges with sufficiently small interior angles, such that for each pair of points (p, q) some double wedge $w(p, q)$ in F has p on one side and q on the other, then every MST edge pq is the bichromatic closest pair for wedge $w(p, q)$.

Suppose the interior angle required is $2\pi/k$. We can divide the space around each point p into k wedges, each having that interior angle. Suppose edge pq falls inside one particular wedge w. We find a collection of double wedges, with sides parallel to w, that is guaranteed to contain pq. By repeating the construction k times, we are guaranteed to find a double wedge containing each possible edge.

For simplicity, assume that the sides of wedge w are horizontal and vertical. In the actual construction, w will have a smaller angle than $\pi/2$, but the details are similar. First choose a horizontal line with at most $n/2$ points above it, and at most $n/2$ points below. We continue recursively with each of these two subsets; therefore if the line does not cross pq, then pq is contained in a closest pair problem generated in one of the two recursive subproblems. At this point we have two sets, above and below the line. We next choose a vertical line, again dividing the point set in half. We continue recursively with the pairs of sets to the left of the line, and to the right of the line. If the line does not cross pq, then pq will be covered by a recursive subproblem. If both lines crossed pq, so that it was not covered by any recursive subproblem, then $w(p, q)$ can be taken to be one of two bichromatic closest pair problems formed by opposite pairs of the quadrants formed by the two lines.

The inner recursion (along the vertical lines) gives rise to one subproblem containing p at each level of the recursion, and each level halves the total number of points, so p ends up involved in one problem of each possible size $n/2^i$. The outer recursion generates an inner recursion at each possible size, giving i problems total of each size $n/2^i$. The construction must be repeated for each of the k wedge angles, multiplying the bounds by $O(1)$. □

Lemma 5 reduces the geometric MST problem to computing a MST in a graph whose vertices are the points of S and whose edges are $O(n \log^{d-1} n)$ bichromatic closest pairs. Insertion or deletion of a point changes $O(\log^d n)$ edges of the graph. Hence, we can maintain the geometric MST by performing $O(\log^d n)$ updates in a data structure for maintaining the MST in a dynamic graph. The following recent result strengthens an $O(m^{1/2})$ time dynamic graph MST algorithm of Frederickson [14].

Lemma 6 (Eppstein *et al.* [12]). *Given a graph subject to edge insertions and deletions, having at most n vertices and m edges at any one time, the minimum spanning tree can be maintained in time $O(n^{1/2} \log(m/n))$ per update.*

If we combine these two results, we get an $O(n^{1/2} \log^d n \log\log n)$ time algorithm for maintaining the MST, once we know the corresponding BCP information. We can save a factor of $O(\log\log n)$ using a more sophisiticated $O(n^{1/2})$-time graph MST algorithm [13] or alternately we can use the following technique. Recall that our collection of BCP problems is formed by splitting a point set recursively with a hyperplane, partitioning the points on each side of the hyperplane recursively, and combining the results with those of a lower-dimensional recursion along the hyperplane itself. For each hyperplane in the recursion, we maintain the minimum spanning tree of the BCP edges defined in all the subproblems both on the hyperplane itself, and in the subproblems on either side of the hyperplane. Each spanning tree in a subproblem with k points is a subgraph of a graph with $3k$ edges, formed as the union of minimum spanning trees in each subproblem. Each update in this spanning tree can be performed in $O(k^{1/2})$ time, and each update in the geometric problem gives rise to $O(\log^d k)$ updates in each subproblem. The total time for performing MST updates is then

$$\sum_{i=0}^{\log_2 n} O(\log^d n) O(n/2^i)^{1/2} = O(n^{1/2} \log^d n).$$

The time bound for maintaining the solutions to the BCP problems is $\sum i^{d-1} T(n/2^i)$ per insertion or deletion, where $T(n)$ is the time to update a single such problem. If $T(n) = \Omega(n^\epsilon)$ for some $\epsilon > 0$ this sum reduces to $O(T(n))$.

The hierarchical structure of Lemma 5 can be periodically rebalanced in a similar amortized time bound. After $\Theta(n/2^i)$ insertions in a subproblem of size $\Theta(n/2^i)$, we rebuild the hierarchical decomposition for that subproblem, reconstruct the BCP data structure, rebuild the MST data structures within that subproblem, and update the MST structures for higher level problems. The time for rebuilding the BCP structure is an amortized $T(n)\log^{O(1)} n$ per operation; again if $T(n) = \Omega(n^\epsilon)$ the polylogarithmic factor goes away. The MST data structure can be rebuilt in close to linear time [12], so it only contributes an additive polylogarithmic term to the total amortized time. We will delete and reinsert $O(n/2^i)$ edges in higher levels of the MST data structure, in time

$$\sum_{j=0}^{i} O(n/2^i)O(n/2^j)^{1/2} = O(n^{3/2}/2^i).$$

So these rebalancing MST updates take an amortized time bound of $O(n^{1/2})$ per insertion to the given subproblem, for a total amortized bound of simply $O(n^{1/2}\log^d n)$.

We have proven our main result:

Theorem 2. *A Euclidean minimum spanning tree of a set of points in R^d can be maintained in amortized time $O(n^{1/2}\log^d n)$ per update for $d \leq 4$ and in time $O(n^{1-2/(\lceil d/2\rceil+1)+\epsilon})$ for $d > 4$.*

Proof: We maintain the hierarchical decomposition into BCP problems of Lemma 5, periodically rebalanced as described above. For each subproblem in the hierarchy we maintain a dynamic graph MST data structure, also described above. Each point insertion or deletion causes BCP updates taking amortized time $O(n^{1-2/(\lceil d/2\rceil+1)+\epsilon})$, MST updates taking time $O(n^{1/2}\log^d n)$, and rebalancing taking amortized time dominated by the BCP and MST updates. □

The space is bounded by that for the nearest neighbor data structure, $O(n^{1+\epsilon})$ in R^2, or worse bounds in higher dimensions.

We note that for rectilinear (L_1 and L_∞) metrics, orthogonal range query data structures can be used to answer dynamic bichromatic closest pair queries in $O(\log^d n)$ time per update [27].

Theorem 3. *The rectilinear MST of a set of n points in R^d can be maintained in time $O(n^{1/2}\log^d n)$ per update.*

7 Conclusions

We have described algorithms for maintaining the minimum spanning tree of a changing point set, the bichromatic closest pair of a colored point set, and many other dynamic geometry problems including diameter and bichromatic farthest pairs.

This naturally raises the question of how much farther our algorithms can be extended or improved. It may be possible to compute the ordered nearest neighbor graph more quickly than the current method, which involves n updates in a post office problem. It is not necessary that the graph form a path; it must only have bounded in-degree.

It is open whether the ordered nearest neighbor sequence we generate, or any other sequence for which the nearest neighbor in-degree is constant, can be found quickly in parallel. Our method of picking a point at a time and finding its nearest neighbor in the remaining set seems inherently sequential, but perhaps a different approach will work.

Acknowledgements

Portions of this paper appear in a preliminary form in a conference abstract [2], co-authored with Pankaj Agarwal and Jirka Matoušek, which also contains results of Agarwal and Matoušek on halfspace range searching and nearest neighbor queries. I thank Pankaj for the many improvements he made in this presentation while readying that paper for publication. This work was supported in part by NSF grant CCR-9258355.

References

[1] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6:407–422, 1991. See also *6th Symp. Comp. Geom.*, 1990, pp. 203–210.

[2] P. K. Agarwal, D. Eppstein, and J. Matoušek. Dynamic algorithms for halfspace reporting, proximity problems, and geometric minimum spanning trees. In *Proc. 33rd IEEE Symp. Foundations of Computer Science*, pages 80–89, 1992.

[3] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. In *Proc. 24th ACM Symp. Theory of Computing*, pages 517–526, 1992.

[4] P. K. Agarwal and M. Sharir. Off-line dynamic maintenance of the width of a planar point set. *Comput. Geom.: Theory & Appl.*, 1:65–78, 1991.

[5] J. L. Bentley and J. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1:301–358, 1980.

[6] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid. Static and dynamic algorithms for k-point clustering problems. In *Proc. 3rd Worksh. Algorithms and Data Structures*, pages 265–276. Lecture Notes in Computer Science 709, Springer-Verlag, Berlin, 1993.

[7] D. Dobkin and S. Suri. Maintenance of geometric extrema. *J. Assoc. Comput. Mach.*, 38:275–298, 1991. See also *30th Symp. Found. Comp. Sci.*, 1989, pp. 488–493.

[8] D. Eppstein. Offline algorithms for dynamic minimum spanning tree problems. In *Proc. 2nd Worksh. Algorithms and Data Structures*, pages 392–399. Springer-Verlag, LNCS 519, 1991.

[9] D. Eppstein. Dynamic three-dimensional linear programming. *ORSA J. Comput.*, 4:360–368, 1992. See also *32nd Symp. Found. Comp. Sci.*, 1991, pp. 488–494.

[10] D. Eppstein. Average case analysis of dynamic geometric optimization. In *Proc. 5th ACM-SIAM Symp. Discrete Algorithms*, pages 77–86, 1994. To appear in *Computational Geometry Theory & Applications.*

[11] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. In *Proc. 4th Symp. Discrete Algorithms*, pages 64–73, 1993.

[12] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification – A technique for speeding up dynamic graph algorithms. In *Proc. 33rd IEEE Symp. Foundations of Computer Science*, pages 60–69, 1992.

[13] D. Eppstein, Z. Galil, and G.F. Italiano. Improved sparsification. Technical Report 93-20, Department of Information and Computer Science, University of California, Irvine, 1993.

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

[15] J. Hershberger and S. Suri. Offline maintenance of planar configurations. In *Proc. 2rd ACM/SIAM Symp. Discrete Algorithms*, pages 32–41, 1991.

[16] G. S. Lueker. A data structure for orthogonal range queries. In *Proc. 19th IEEE Symp. Foundations of Computer Science*, pages 28–34, 1978.

[17] G. S. Lueker and D. E. Willard. A data structure for dynamic range queries. *Inform. Proc. Lett.*, 15:209–213, 1982.

[18] J. Matoušek and O. Schwarzkopf. Linear optimization queries. In *Proc. 8th ACM Symp. on Computational Geometry*, pages 16–25, 1992.

[19] K. Mulmuley. Randomized multidimensional search trees: dynamic sampling. In *Proc. 7th ACM Symp. Computational Geometry*, pages 121–131, 1991.

[20] K. Mulmuley. Randomized multidimensional search trees: lazy balancing and dynamic shuffling. In *Proc. 32nd IEEE Symp. Foundations of Computer Science*, pages 180–196, 1991.

[21] K. Mulmuley. *Computational Geometry, an Introduction through Randomized Algorithms*. Prentice Hall, 1993.

[22] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comp. Syst. Sci.*, 23:224–233, 1981.

[23] C. Schwarz, M. Smid, and J. Snoeyink. An optimal algorithm for the on-line closest pair problem. In *Proc. 8th ACM Symp. Computational Geometry*, pages 330–336, 1992.

[24] O. Schwarzkopf. Dynamic maintenance of geometric structures made easy. In *Proc. 32nd IEEE Symp. Foundations of Computer Science*, pages 197–206, 1991.

[25] M. Smid. Maintaining the minimal distance of a point set in polylogarithmic time. *Discrete Comput. Geom.*, 7:415–431, 1992. See also *2nd Symp. Discrete Algorithms*, pages 1–6, 1991.

[26] K. J. Supowit. New techniques for some dynamic closest-point and farthest-point problems. In *Proc. 1st ACM-SIAM Symp. Discrete Algorithms*, pages 84–90, 1990.

[27] P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18:1201–1225, 1989.

[28] D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *J. Assoc. Comput. Mach.*, 32:597–617, 1985.