Dynamic Half-Space Reporting, Geometric Optimization, and Minimum Spanning Trees

Pankaj K. Agarwal* David Eppstein† Jiří Matoušek‡

Abstract

We describe dynamic data structures for half-space range reporting and for maintaining the minima of a decomposable function. Using these data structures, we obtain efficient dynamic algorithms for a number of geometric problems, including closest/farthest neighbor searching, fixed dimension linear programming, bi-chromatic closest pair, diameter, and Euclidean minimum spanning tree.

1 Introduction

Dynamic data structures involve updating the solution to a problem (either explicitly or implicitly, depending on the problem) as the set of input objects changes dynamically. The study of dynamic algorithms has been motivated by applications in several applied areas including optimization, computer graphics, and VLSI. Also, solutions to several static problems require efficient dynamic algorithms either for a similar problem in lower dimensions (e.g. point-location [31, 30] or computing the area of rectangles [29]), or for a different problem (e.g., convex layers [6] or hidden surface removal [4]). As a result, a lot of attention has been paid to studying dynamic geometric algorithms. For some examples see [3, 4, 16, 18, 20, 23, 28, 32, 33, 34], or see [11] for an excellent survey on this subject. Researchers have also studied the special cases when objects are only allowed to be inserted (deletions are not allowed), or when the sequence of insertions and deletions is known in advance. If the problem is decomposable (i.e., the solutions in subsets of the input can be efficiently combined to determine the overall solution) then one can obtain fast dynamic algorithms for these special cases [5, 17]. Recently Dobkin and Suri extended these techniques to a *semi-online* model in which insertions are completely arbitrary, but when an object is inserted we are told its deletion time [16]. However, these techniques do not seem to extend to the general case where deletions are also arbitrary.

In this paper we present efficient solutions to two dynamic geometric problems: halfspace range reporting and finding the minima of a decomposable function. Using these results, we develop new algorithms for a wide variety of fully dynamic geometric optimization problems, including maintenance of a minimum spanning tree of a point set. Our algorithms are either the first known solutions to these problems, or they are significantly faster than previously known algorithms. The following list describes the problems considered here and Table 1 summarizes the results presented in this abstract.

Half-space range reporting: The first result of this paper is a dynamic data structure for half-space range reporting, which can be formally defined as follows: Store a set S of points in $\mathbb{R}^d$ into a data structure, so that all k points of S lying in a query half-space can be reported quickly.[1] In several applications it suffices to determine whether the query half-space contains any point of S. We will refer to this restricted version as the *empty half-space problem*. Half-space range reporting is a special case of the general simplex range searching problem in which one wants to report all points of S lying in a query simplex. Using an algorithm due to Clarkson, one can store S in a data structure of size $O(n^{\lfloor d/2 \rfloor + \varepsilon})$ so that a half-space query can be answered in time $O(\log n + k)$ [12]. Recently Matoušek showed that a half-space query can be answered in time $O(n^{1-1/\lfloor d/2 \rfloor} \log^{O(1)} n + k)$ using $O(n \log n)$ preprocessing time and $O(n \log \log n)$ space [24]. Combining these two approaches, for a

*Computer Science Department, Duke University, Durham, NC 27706. Work supported by National Science Foundation Grant CCR-91-06514.

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

‡Department of Applied Mathematics, Charles University, Prague, and Freie Universität Berlin.

[1] Throughout this paper we assume d to be some fixed positive integer and ε to be an arbitrarily small constant. The constants in the time complexity of the algorithms depend on d and ε.

given parameter m, $n \leq m \leq n^{\lfloor d/2 \rfloor}$, one can answer a half-space query in time $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log n + k)$ using $O(m^{1+\varepsilon})$ space and preprocessing.

Known simplex range searching data structures can be dynamized easily. In the plane, dynamic half-space range searching can be performed using the dynamic convex hull structure of Overmars and van Leeuwen [28]. But no efficient algorithm was known for higher dimensional half-space searching. We present a data structure that can answer a half-space range reporting query in time $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log n + k)$, and can insert or delete a point in amortized time $O(m^{1+\varepsilon}/n)$. It can also answer an empty half-space query in time $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log n)$ (cf. Section 2).

Minima of decomposable functions: Our second result is an algorithm for maintaining the minima of a decomposable function over a set of points in $\mathbb{R}^d$. Let R be a set of red points and let B be a set of blue points. For a given distance function $\delta(.,.)$, we define $\delta(R, B) = \min_{p \in R, q \in B} \delta(p, q)$. The goal is to maintain $\delta(R, B)$ as points are inserted to or deleted from $R \cup B$. This problem can also be viewed as computing a shortest edge in the complete weighted bipartite graph $R \times B$. One can also compute maxima with the same algorithm by negating function $\delta(.,.)$.

This problem in its full generality was first studied by Dobkin and Suri [16]. They showed that the minima can be maintained efficiently in the semi-online model. But no fully dynamic algorithm was known for this problem. We show that if we have a dynamic data structure that can compute a nearest neighbor for a query point with respect to $\delta(.,.)$, then $\delta(R, B)$ can be maintained in amortized time $O(T(n) \log^2 n)$, where $n = |R| + |B|$ and $T(n)$ is the query/update time of the nearest neighbor searching structure (cf. Section 3).

In [2] several geometric problems have been reduced to answering empty half-space queries, and in [16] a number of problems have been reduced to computing the minima (or maxima) of a decomposable function. Our algorithms yield efficient dynamic solutions to these problems too. We mention some of them here.

Ray shooting in a convex polyhedron: Given a convex polyhedron P in $\mathbb{R}^d$ defined as the intersection of n half-spaces, preprocess it, so that the first intersection point of P and a query ray can be computed quickly. Agarwal and Matoušek [2] showed that such a query can be answered using a data structure for the half-space emptiness queries (satisfying certain mild requirements, which are satisfied by the data structure we construct), with some polylogarithmic overhead in query time. Hence, the above query can be answered in time $O((n \log^3 n)/m^{\lfloor d/2 \rfloor})$ and a half-space can be inserted or deleted in time $O(m^{1+\varepsilon}/n)$. They also showed that dynamic ray shooting procedure can be used to compute convex layers and higher order Voronoi diagrams. Combining these reductions with our results, we can compute the convex layers of n points in $\mathbb{R}^3$ in time $O(n^{1+\varepsilon})$, and we can compute the k^{th}-order Voronoi diagram of n points in the plane in time $O(n^{1+\varepsilon}k)$.

Closest/farthest neighbor queries: Given a set S of n points in $\mathbb{R}^d$, maintain a data structure so that the closest (or farthest) neighbor of a query point in S can be computed efficiently. Although there are efficient dynamic algorithms for maintaining the closest distance in S, no such procedure was known for computing the closest neighbor of a query point in S. It is shown in [2] that this problem can be reduced to answering a ray shooting query in a convex polyhedron in $\mathbb{R}^{d+1}$ defined by n half-spaces, which implies that the closest/farthest neighbor queries can be answered in time $O((n \log^3 n)/m^{1/\lceil d/2 \rceil})$ and a point can be inserted to or deleted from S in amortized time $O(m^{1+\varepsilon}/n)$. In particular, closest/farthest neighbor queries in $\mathbb{R}^2$ can be answered in time $O(\log^3 n)$, and a point can be inserted to or deleted from S in amortized time $O(n^{\varepsilon})$. The previously best known algorithm took time $O(\sqrt{n \log n})$ per update or query.

Bi-chromatic closest pair and diameter: Given a set R red points and a set B of blue points, determine the closest or farthest red-blue pair. The best known algorithm for computing the bi-chromatic closest pair, due to Agarwal et al. [1], has time complexity $O(n^{2(1-1/(\lceil d/2 \rceil+1))+\varepsilon})$. Recently Chazelle et al. [7] showed that a bi-chromatic farthest pair can be computed in time $O(n^{2(1-1/(\lfloor d/2 \rfloor+1))+\varepsilon})$. The diameter can be computed using this algorithm by setting $R = B = S$.

Dobkin and Suri [16] gave efficient solutions to these problems in the semi-online model. Vaidya [34] described an algorithm for closest pairs which allows insertions and deletions to R, but only allows insertions in B. Supowit [33] showed that if we allow only deletions, then the diameter of a planar point set can be maintained in $O(\sqrt{n} \log n)$ amortized time. But even in two dimensions, no fully dynamic algorithm was known for these problems. Using our algorithms for nearest neighbors and for minima of decomposable functions, we present a data structure that can maintain the bi-chromatic closest or farthest pair, or the diameter, in time $O(n^{1-2/(\lceil d/2 \rceil+1)+\varepsilon})$ per update. In particular, this is $O(n^{\varepsilon})$ for planar point sets.

Linear Programming: Store a set of linear con-

Problem	Query time	Update time
Empty half-space	$O(\frac{n \log n}{m^{1/\lfloor d/2 \rfloor}})$	$O(\frac{m^{1+\varepsilon}}{n})$
Half-space reporting	$O(\frac{n \log n}{m^{1/\lfloor d/2 \rfloor}} + k)$	$O(\frac{m^{1+\varepsilon}}{n})$
Ray shooting in convex polyhedron	$O(\frac{n \log^3 n}{m^{1/\lfloor d/2 \rfloor}})$	$O(\frac{m^{1+\varepsilon}}{n})$
Closest/farthest neighbor	$O(\frac{n \log^3 n}{m^{1/\lceil d/2 \rceil}})$	$O(\frac{m^{1+\varepsilon}}{n})$
Linear programming	$O(\frac{n \log^{O(1)} n}{m^{1/\lceil d/2 \rceil}})$	$O(\frac{m^{1+\varepsilon}}{n})$
Bi-chromatic closest/farthest pair		$O(n^{1-2/(\lceil d/2 \rceil+1)+\varepsilon})$
Diameter		$O(n^{1-2/(\lceil d/2 \rceil+1)+\varepsilon})$
MST, L_1 or L_∞ metric		$O(\sqrt{n} \log^d n)$
EMST, $d \leq 4$		$O(\sqrt{n} \log^d n)$
EMST, $d > 4$		$O(n^{1-2/(\lceil d/2 \rceil+1)+\varepsilon})$

Table 1: Summary of results

straints (half-spaces) in $\mathbb{R}^d$ into a data structure, so that one can find a point that minimizes a query objective function. This problem had been previously been solved in $O(\log^2 n)$ time per operation in two dimensions [28] and $O(\sqrt{n} \log n)$ randomized expected time in three dimensions [19]. If we use Megiddo's parametric search, the problem can be solved using a suitable data structure empty for half-space queries [25], which implies that we can answer queries in time $O((n \log^{O(1)} n)/m^{1/\lfloor d/2 \rfloor})$, and can insert or delete a constraint in $O(m^{1+\varepsilon}/n)$ amortized time.

Smallest enclosing disk maintenance. A corollary of the previous result is that a smallest enclosing disk for an n point set in $\mathbb{R}^d$ can be maintained in amortized $O(n^{1-\frac{1}{\lceil d/2 \rceil+1}+\varepsilon})$ time per insertion or deletion of a point of S [26].

Euclidean minimum spanning tree: Given a set S of n points in $\mathbb{R}^d$, compute a Euclidean minimum spanning tree of S. For $d = 2$, an optimal $O(n \log n)$ static algorithm has long been known. For $d \geq 3$ the best known algorithm, due to Agarwal et al. [1], has time complexity $O(n^{2(1-1/(\lceil d/2 \rceil+1))+\varepsilon})$. Point insertions are not difficult to handle, and an offline algorithm for the planar EMST with insertions and deletions is described in [18]. But no fully dynamic algorithm was known. We use a reduction to the bichromatic closest pair problem due to Agarwal et al. together with a modification of the new dynamic graph MST algorithm of Eppstein et al. [21] to show that the EMST of S can be maintained in $O(\sqrt{n} \log^d n)$ amortized time for $d \leq 4$ and in amortized time $O(n^{1-2/(\lceil d/2 \rceil+1)+\varepsilon})$ for $d > 4$ (cf. Section 4).

2 Half-Space Range Reporting

In this section we describe an algorithm for dynamic half-space range reporting. The main result of this section is

Theorem 2.1 *Given a set S of n points in $\mathbb{R}^d$ ($d \geq 3$), the half-space range reporting problem can be solved with the following performance:*

(i) *$O(n \log n)$ space and preprocessing time, $O(n^{1-1/\lfloor d/2 \rfloor+\varepsilon}+k)$ time per query, and $O(\log^2 n)$ amortized update time*

(ii) *$O(n^{\lfloor d/2 \rfloor+\varepsilon})$ space and preprocessing , $O(\log n+k)$ query time and $O(n^{\lfloor d/2 \rfloor-1+\varepsilon})$ amortized update time, and*

(iii) *For a parameter m between n and $n^{\lfloor d/2 \rfloor}$, $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log n)$ query time, $O(m^{1+\varepsilon})$ space and preprocessing time, and $O(m^{1+\varepsilon}/n)$ amortized update time.*

Due to lack of space, we will prove only the second (most difficult) part of the above theorem. We will follow an approach similar to that of Chazelle et al. [10]. Since the half-space range reporting problem is decomposable, i.e., the answers for two disjoint point sets S_1 and S_2 can be combined into an answer for $S_1 \cup S_2$ in constant time, it suffices to describe a data structure that supports only delete operations. Such a data structure can be modified to handle insertions as well; e.g., see [27].

We first describe a data structure for the half-space emptiness problem (i.e., given a query half-space determine whether it contains any point of S), and then

extend it to the half-space reporting problem. Let H be the set of n hyperplanes dual to the points in S. Answering an empty half-space query for S reduces to determining whether a query point p lies above all hyperplanes of H; we refer to the dual problem as the *upper envelope problem* for H. We describe our data structure in this dual formulation.

We need the following results of [24]. The $(\leq k)$-level of H is the set of points $p \in \mathbb{R}^d$ such that at most k hyperplanes of H lie (strictly) above p. For a parameter $r \leq n$, we define a $(1/r)$-cutting of $(\leq k)$-level of H to be a set Ξ of pairwise disjoint simplices such that Ξ covers the $(\leq k)$-level of H and that each simplex of Ξ intersects at most n/r hyperplanes of H. We say that a hyperplane h is *relevant* for a simplex $\triangle$ if h lies above $\triangle$ or intersects $\triangle$. Let $H_\triangle \subseteq H$ denote the set of hyperplanes relevant for a simplex $\triangle \in \Xi$. If $|H_\triangle| > k + n/r$, then $\triangle$ cannot contain any point of $(\leq k)$-level of H, so we can drop $\triangle$ from Ξ. Henceforth we assume that $|H_\triangle| \leq k + n/r$.

Theorem 2.2 (Matoušek [24]) *Let H, k, and r be as above, with $k = O(n/r)$. Then there exists a $(1/r)$-cutting Ξ for the $(\leq k)$-level of H, consisting of $s(r) = O(r^{\lfloor d/2 \rfloor})$ simplices. For $r \leq n^\alpha$ (where $\alpha > 0$ is a constant depending on d), such a cutting can be computed in $O(n \log r)$ time.*

2.1 The data structure

Let δ be a small positive constant fixed throughout the construction (the ε in the resulting performance bounds as well as the constants hidden in the asymptotic notation will depend on δ). The data structure will be periodically rebuilt from scratch after deleting some of the hyperplanes present at the moment of the previous global reconstruction. We use m to denote the number of hyperplanes in H when the data structure was constructed last time, and n to denote the number of hyperplanes in the current H. We set $r = m^\delta$; this setting remains valid between global reconstructions.

We reconstruct the data structure from scratch after deleting $m/2r$ hyperplanes from H. Thus $n \geq m(1 - 1/2r)$. Let $P(m)$ denote the time spent in preprocessing H. The time spent for these global reconstructions can be amortized by charging $2rP(m - \frac{m}{2r})/m \leq 2rP(n)/n$ time to each delete operation. As we will see, this contribution to the amortized deletion time will not affect the asymptotic performance of the structure.

The data structure for H is a recursively defined tree, denoted by $\Psi(H)$. The substructures of $\Psi(H)$ will be periodically reconstructed during the deletions, but this reconstruction will not change the value of r. Let us describe how a subtree for a subset $G \subseteq H$, denoted by $\Psi(G)$, is constructed: If $\nu = |G| \leq r$, then $\Psi(G)$ is a leaf node. We preprocess G in time $O(r^{\lfloor d/2 \rfloor + \varepsilon})$ for the upper envelope problem using Clarkson's static data structure. Thus, for a leaf node, the upper envelope problem for G can be solved in $O(\log r)$ time.

Let us now assume $\nu > r$. The root of the tree $\Psi(G)$ will store the following items:

- A partition of G into disjoint subsets $G_1, G_2, \ldots, G_t$.
- For every $i = 1, 2, \ldots, t$, a cutting Ξ_i and a point location structure for Ξ_i; see below for details.
- For every $i = 1, 2, \ldots, t$ and every simplex $\triangle \in \Xi_i$, a pointer to a subtree of the form $\Psi(G_{i,\triangle})$, where $G_{i,\triangle}$ is the subset of hyperplanes of G_i relevant for $\triangle$.
- For each hyperplane $h \in G_i$, the list L_h of simplices $\triangle \in \Xi_i$ for which h is relevant.
- A counter *dcount* (used for the deletion algorithm).

After the construction of $\Psi(G)$ (before any deletions takes place), these objects will have the following properties:

1. The counter *dcount* is set to $\nu/2r$.
2. For every i, the simplices of Ξ_i cover all points of level at most ν/r with respect to G.
3. For every i and $\triangle \in \Xi_i$, $|G_{i,\triangle}| \leq 2\nu/r$.
4. For every hyperplane $h \in G_i$, h intersects at most
$$\kappa = c_1 \cdot r^{\lfloor d/2 \rfloor - 1 + \delta}$$
simplices of Ξ_i, where c_1 is some constant.

Property 3 implies that the depth of $\Psi(G)$ is $O(\log_r \nu) = O(1/\delta)$.

The construction of these objects proceeds by induction. Suppose that the collections $G_1, G_2, \ldots, G_{i-1} \subseteq G$ have already been constructed. Let $\bar{G}_i = G \setminus (G_1 \cup \ldots \cup G_{i-1})$, $\nu_i = |\bar{G}_i|$. Let $r_i = r\nu_i/\nu$ and $k = \nu/r = \nu_i/r_i$. We compute a $(1/r_i)$-cutting Ξ_i of size $O(r_i^{\lfloor d/2 \rfloor})$ for the $(\leq k)$-level of $\bar{G}_i$, as mentioned in Theorem 2.2, and preprocess Ξ_i for point location queries as follows. We extend the simplices of Ξ to hyperplanes and preprocess the set

of resulting hyperplanes using the algorithm of Clarkson [14]. The space and preprocessing required by this structure is $r^{O(1)}$ and the query time is $O(\log r)$.

For every $\triangle \in \Xi_i$, let $\bar{G}_{i,\triangle}$ denote the collection of hyperplanes of $\bar{G}_i$ relevant for $\triangle$ (recall that a hyperplane is relevant for $\triangle$ if its passes above or through $\triangle$). As explained above, we assume

$$|\bar{G}_{i,\triangle}| \le \frac{\nu_i}{r_i} + k = \frac{2\nu_i}{r_i} = \frac{2\nu}{r}.$$

A hyperplane $h \in \bar{G}_i$ is called *good* if it is relevant for at most κ hyperplanes of $\bar{G}_i$, and *bad* otherwise. Since

$$\sum_{\triangle\in\Xi_i} |\bar{G}_{i,\triangle}| \le O(r_i^{\lfloor d/2\rfloor}) \cdot \frac{2\nu_i}{r_i} = c_1\nu_i \left(\frac{\nu_i r}{\nu}\right)^{\lfloor d/2\rfloor - 1}$$

(c_1 is a constant appearing in the bound on the size of Ξ_i), the number of 'bad' hyperplanes in $\bar{G}_i$ is at most

$$\left(\sum_{\triangle\in\Xi_i} |\bar{G}_{i,\triangle}|\right) / \kappa \;\le\; \frac{\nu_i}{r^\delta} \cdot \left(\frac{\nu_i}{\nu}\right)^{\lfloor d/2\rfloor - 1} \le \quad \frac{\nu_i}{r^\delta}\,.$$

Let G_i be the set of good hyperplanes in $\bar{G}_i$ and $\bar{G}_{i+1} = \bar{G}_i - G_i$.

If $|\bar{G}_{i+1}| > \nu/r$, we continue the above described construction inductively for $i+1$. Otherwise Ξ_{i+1} consists of a sufficiently large simplex and $G_{i+1} = \bar{G}_{i+1}$. This finishes the construction of the objects stored at the root of $\Psi(G)$. The appropriate subtrees $\Psi(G_{i,\triangle})$ are constructed recursively. The above construction guarantees $\nu_i/\nu_{i+1} \ge r^\delta$, and it finishes when $\nu_i \le \nu/r$, therefore $t \le 1/\delta$. It can be shown that the properties 1–4 are guaranteed by the construction.

Let us estimate the space $S(\nu)$ needed for $\Psi(G)$. If $\nu \le r$, then the space required is $O(r^{\lfloor d/2\rfloor+\varepsilon})$. Otherwise, we need $O(tr^c)$ space to store the point location structures for $\Xi_1, \ldots, \Xi_t$ and $O(t\nu r^{\lfloor d/2\rfloor-1})$ space to store the lists L_h for each $h \in G$. Since $|G_{i,\triangle}| \le 2\nu/r$, we get the following recurrence:

$$S(\nu) = \begin{cases} O(r^{\lfloor d/2\rfloor+\varepsilon}) & \nu \le r \\ O(\nu r^{\lfloor d/2\rfloor-1} + r^c) + \displaystyle\sum_{i\le t,\ \triangle\in\Xi_i} S(\frac{2\nu}{r}) & \nu > r\,. \end{cases}$$

Since each Ξ_i consists of $O(r^{\lfloor d/2\rfloor})$ simplices and t is bounded by a constant, the solution of the above recurrence is $S(\nu) = O((r\nu)^{\lfloor d/2\rfloor+\varepsilon})$. By a similar computation, we get $P(\nu) = O((r\nu)^{\lfloor d/2\rfloor+\varepsilon})$ for the time required for building $\Psi(G)$.

2.2 Answering a query

We answer an upper envelope query for a query point p recursively. Suppose we are at the root v of a subtree $\Psi(G)$. If v is a leaf node, we answer a query in $O(\log r)$ time using Clarkson's structure stored at v. Otherwise, p lies in the upper envelope of G if and only if for each $i = 1, 2, \ldots, t$ it lies in the upper envelope of G_i. To test this, first determine the simplex $\triangle \in \Xi_i$ that contains p. If there is no such simplex, p does not lie in the upper envelope of G_i. Otherwise, recursively answer the query in $\Psi(G_{i,\triangle})$.

The depth of the overall tree $\Psi(H)$ is $O(1/\delta)$ and the branching degree in this query answering process is always at most $t = O(1)$, which implies that only $O(1)$ nodes are visited. Since we spend $O(\log r)$ time at each node, the total query time is $O(\log n)$.

2.3 Deleting a hyperplane

Let us describe the algorithm for deleting a hyperplane h from H. We visit $\Psi(H)$ in a top-down fashion, and at the root v of each subtree $\Psi(G)$ visited, we do the following: If v is a leaf, we rebuild the structure stored at v. If v is not a leaf, we first decrement the counter *dcount* stored at v by one. If *dcount* becomes zero, we rebuild $\Psi(G)$ (for the current G). Otherwise, we find the i with $h \in G_i$. We delete h from G_i, and then recursively delete h from all subtrees of the form $\Psi(G_{i,\triangle})$ with $\triangle \in L_h$. By construction, $|L_h| \le \kappa$.

The deletion algorithm guarantees that the properties 3 and 4 always hold, and 2 is replaced by a weaker one:

> $2'$. For each i, the simplices of Ξ_i cover all points of level at most $\nu/2r$ with respect to the current set G.

The correctness of the query answering algorithm follows immediately from $2'$ (actually even a weaker version of $2'$, where all points of level 0 are covered, would suffice; this stronger form anticipates the extension to half-space range reporting) and the above discussion.

In order to finish the proof of Theorem 2.1(ii), it suffices to estimate the amortized deletion time. To this end, let $D(\nu, k)$ denote the maximum time needed for deleting k hyperplanes from the data structure $\Psi(G)$ with $|G| = \nu$, starting at the moment $\Psi(G)$ was built anew. As noted above, a deletion of one hyperplane from G propagates into at most κ children of the root of $\Psi(G)$, thus before the next complete reconstruction of $\Psi(G)$, there are at most $(\nu/2r)\kappa$ deletions

in the children. We thus get the following recurrences:

$$D(\nu,k) \;\le\; \begin{cases} O(kr^{\lfloor d/2\rfloor+\varepsilon}) & \text{for } \nu \le r \\ D(\nu,\frac{\nu}{2r}) + D(\nu-\frac{\nu}{2r}, k-\frac{\nu}{2r}) + \\ P(\nu-\frac{\nu}{2r}), & \text{for } \nu > r,\ k > \nu/2r \\ \max\{\sum_j D(\frac{2\nu}{r}, k_j) \mid \sum_j k_j \le k\kappa\} + \\ O(k\log\nu) & \text{for } \nu > r,\ k \le \nu/2r \end{cases}$$

The solution of the above recurrence is

$$D(\nu,k) \le C^{\log_r \nu} k\nu^{\lfloor d/2\rfloor-1+\delta} r^{\lfloor d/2\rfloor+\varepsilon},$$

for some large enough constant C. In particular, we get $D(m,m) = O(m^{\lfloor d/2\rfloor+c_2\delta})$, where the constant c_2 is independent of δ.

Hence we obtain a data structure that, using $O(n^{\lfloor d/2\rfloor+\varepsilon})$ space and preprocessing, can answer an empty half-space query in time $O(\log n)$ and can delete a point in $O(n^{\lfloor d/2\rfloor-1+\varepsilon})$ amortized time. In view of the discussion in the beginning, we obtain

Theorem 2.3 *Given a set S of n points in $\mathbb{R}^d$, we can preprocess it in $O(n^{\lfloor d/2\rfloor+\varepsilon})$ time and space, so that an empty half-space query can be answered in $O(\log n)$ time, and a point can be inserted to or deleted from S in amortized time $O(n^{\lfloor d/2\rfloor-1+\varepsilon})$.*

2.4 Half-space reporting

We will now briefly describe how to modify the above procedure for half-space reporting (actually its dual version — reporting hyperplanes of H lying above a query point). Let v be a node in the above described data structure for half-space emptiness queries, and let G be the corresponding set of hyperplanes. We preprocess G for half-space reporting using Clarkson's static data structure (using space and preprocessing time $O(m^{\lfloor d/2\rfloor+\varepsilon})$) and store this secondary structure at v.

When we reconstruct $\Psi(G)$, we also reconstruct the secondary structure with the current G. But when we delete a hyperplane h we just update the primary structure as described earlier. The time spent in deleting a hyperplane obviously remains the same. Notice that we are updating the secondary structure only when we reconstruct $\Psi(G)$, so the secondary structure will store $\tilde{G}$, the set of hyperplanes in G when $\Psi(G)$ was constructed last time.

As for answering a query, if v is a leaf, we report, in time $O(\log\nu + k_v) = O(\log r + k_v)$, all k_v hyperplanes of G lying above p using the structure stored at v. If v is not a leaf, we determine in time $O(\log r)$ the simplex $\triangle_i \in \Xi_i$ $(i \le t)$ that contains p. If $\triangle_i$ is defined for all $i \le t$, then we recursively search in $\Psi(G_{i,\triangle_i})$ with p. Otherwise there is some i such that p does not lie in any simplex of Ξ_i, which, by condition 2′, implies that there are at least $\nu/2r$ (current) hyperplanes above p. In this case, we query the secondary structure stored at v and compute in time $O(\log\nu + \tilde{k}_v)$ all $\tilde{k}_v$ hyperplanes of $\tilde{G}$ that lie above p. We then report the subset of these hyperplanes that have not been deleted. Since $\tilde{k}_v \le k_v + \nu/2r$ and $k_v \ge \nu/2r$, we have $_v \le 2k_v$ and the overall query time is thus $O(\log n + k)$. This finishes the proof of Theorem 2.1 (ii).

An immediate corollary of Theorem 2.1 is

Corollary 2.4 *Given a set S of n points in $\mathbb{R}^d$ and a parameter $n \le m \le n^{\lceil d/2\rceil}$, we can preprocess it in $O(m^{1+\varepsilon})$ time and space, so that a closest neighbor of a query point in S can be computed in $O((n\log^3 n)/m^{1/\lceil d/2\rceil})$ time, and a point can be inserted to or deleted from S in amortized time $O(m^{1+\varepsilon}/n)$.*

3 Minima of Decomposable Functions

We now describe a data structure for maintaining minima of a decomposable function defined over a set of points. Let R be a set of m red points and B a set of n blue points, and let $\delta(.,.)$ be a distance function. (Note that δ need not satisfy the triangle inequality, so it may not be a metric.) We define $\delta(R,B) = \min\delta(p,q)$, where p varies over all points of R and q varies over all points of B. The goal is to maintain $\delta(R,B)$ as points are inserted to or deleted from $S = R \cup B$. One can maintain maxima instead of minima by negating δ. We first introduce the notion of ordered nearest neighbors and then describe our algorithm.

3.1 Ordered nearest neighbors

Let $x_1, x_2, \ldots, x_n$ be a sequence of points, each of which is colored red or blue. We define the *bi-chromatic ordered nearest neighbor* of a red (resp. blue) point x_i to be a blue (resp. red) point x such that $\delta(x,x_i)$ is minimized over all blue (resp. red) points in the set $\{x_j : j > i\}$. If all $x_i, \ldots, x_n$ have the same color, then their ordered bi-chromatic closest neighbors are not defined. The *ordered bi-chromatic nearest neighbor graph* is a directed graph in which each point has an edge directed to its bi-chromatic ordered nearest neighbor. Suppose we have a dynamic data

structure for answering nearest neighbor queries with respect to $\delta(.,.)$, with preprocessing time $P(n)$ and query and update time $T(n)$.

Lemma 3.1 *Let S be any bi-chromatic point set with m red points and $n > m$ blue points. Then the points of S can be ordered into a sequence, in time $O(P(n) + mT(n))$, such that the ordered bi-chromatic nearest neighbor graph forms a simple path.*

Proof: Choose x_1 arbitrarily. Assume inductively that we have chosen x_1 through x_i so the nearest neighbor of each point x_j $(1 \leq j < i)$, among points later in the sequence or not yet chosen, is x_{j+1}. To extend this sequence by one more point, choose x_{i+1} to be the nearest neighbor to x_i among the unchosen points. The resulting path will have length at most $2m$, after which we can place the remaining $n - 2m$ points in any order. We maintain a nearest neighbor searching structure for computing a nearest neighbor among the unchosen points. When we choose a point we remove it from the structure. There are at most $2m$ queries and a similar number of deletions, so the total time is $O(P(n) + mT(n))$. □

The $O(P(n))$ term is a one time start up cost for building the nearest neighbor searching data structure. Once we have computed the ordered bi-chromatic nearest neighbor graph, we can add back the deleted points in time $O(mT(n))$, matching the time for constructing the graph. Then we can remember the data structure and re-use it in later computations, avoiding the start-up cost. If a point is inserted to or deleted from S, the nearest neighbor searching structure can be updated in time $O(T(n))$.

3.2 The data structure

We partition the point set S into *levels* numbered from 0 to $O(\log n)$. 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. For each S_i we maintain a graph G_i.

As we will show below, one of these graphs will contain an edge (p, q) such that $\delta(p, q) = \delta(R, B)$, and it can be found by storing all edges of these graphs in a priority queue. 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 $B_i \cup R$. G_i can be constructed in time $O(|S_i|T(n))$ by Lemma 3.1 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. The amortized time incurred by this reconstruction can be made worst case per operation by a standard trick of keeping two copies of the data structure, one of which is gradually reconstructed while the other is in use.

3.3 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.

Next, we delete a point q from S as follows. We delete all the edges incident to q from each G_i. If $q \in S_i$, then there are at most four edges incident to q, otherwise there are at most two such edges. 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. Since there are most two edges of the above form in each G_i, only $O(\log n)$ 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.

3.4 Correctness

In order to prove the correctness of the algorithm, we need the following simple lemma.

Lemma 3.2 *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 3.3 *There is an edge (p, q) in one of the graphs G_i such that $\delta(p, q) = \delta(R, B)$.*

Proof: Let (p, q) be a bi-chromatic closest pair in S. Suppose that $p \in R_i$ and $q \in B_j$, and that $j \geq i$. It follows from Lemma 3.2 that $q \in S$ when G_i was constructed the last time. First assume that $q < p$ in the ordering for which we have defined the ordered nearest neighbor graph for $R_i \cup B$. Let (q, r) be the edge in G_i when it was constructed the last time. By definition of the ordered nearest neighbor graph, $\delta(q, r) \leq \delta(q, p)$ since $q < p$. If the edge (q, r) still exists, then the lemma is obviously true, so assume that (q, r) has been deleted. 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 3.2, implies that $j < i$, a contradiction. A similar contradiction can be obtained if $p < q$. Hence, the lemma is true. □

Thus $\delta(R, B)$ can be found by maintaing the edges of each G_i in a priority queue.

3.5 Time analysis

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

$$\Phi(p) = cT(n)(k - i)\,,$$

where $k = O(\log n)$ is the maximum number of levels and c is some appropriate constant. We define the overall potential function $\Phi(S) = \sum_{p \in S} \Phi(p)$. First, 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 [15]. But observe that G_i is constructed only when the 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 new point, we add it to S_0, which increases $\Phi(S)$ by $cT(n)$. 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 $O(\log n) \cdot ckT(n) = O(T(n) \log^2 n)$. 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 3.4 *Let $\delta(.,.)$ be a distance function for which we can perform point queries, and insert and delete points, in time $O(T(n))$ per operation. Then we can maintain $\delta(R, B)$ in amortized time $O(T(n) \log n)$ per point insertion, and $O(T(n) \log^2 n)$ per deletion.*

The above theorem and Corollary 2.4 imply that

Corollary 3.5 *A Euclidean bi-chromatic closest pair or farthest pair of a set of n points in $\mathbb{R}^d$, as well as the diameter of such a set, can be maintained in amortized time $O(n^{1-2/(\lceil d/2 \rceil + 1) + \varepsilon})$ per update.*

These techniques also yield efficient fully dynamic algorithms for maintaining the minimum separation among rectangles or higher-dimensional orthogonal boxes, the minimum or maximum distance between points and hyperplanes, and the minimum or maximum box defined by a set of points.

4 Euclidean Minimum Spanning Trees

We have seen how to use the nearest neighbor searching problem to maintain the bi-chromatic 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 bi-chromatic closest pairs and minimum spanning trees can be seen from the following lemma.

Lemma 4.1 (Agarwal et al. [1]) *Given a set of n points in $\mathbb{R}^d$, we can form a hierarchical collection of $O(n \log^{d-1} n)$ bi-chromatic 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.*

Lemma 4.1 reduces the geometric MST problem to computing a MST in a graph whose vertices are the points of S and whose edges are $O(n \log^{d-1} n)$ bi-chromatic 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 an algorithm to maintain the MST in a dynamic graph. The following recent result strengthens an $O(\sqrt{m})$ time dynamic graph MST algorithm of Frederickson [22].

Lemma 4.2 (Eppstein et al. [21]) *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 $O(\sqrt{n}\log(m/n))$ per update.*

If we combine these two results, we get an $O(\sqrt{n}\log^d n \log\log n)$ time algorithm for maintaining the MST, once we know the corresponding BCP information. We can save a further factor of $O(\log\log n)$ by maintaining MSTs of subproblems defined by the hierarchical structure of Lemma 4.1, in a similar fashion to the way Eppstein et al. prove Lemma 4.2 using a hierarchical partition of the graph. The time for solving the BCP problems reduces to a geometric series which is bounded by the time for a single problem. The hierarchical structure of Lemma 4.1 can be periodically rebalanced in a similar amortized time bound. Thus we have

Theorem 4.3 *A Euclidean minimum spanning tree of a set of points in $\mathbb{R}^d$ can be maintained in amortized time $O(\sqrt{n}\log^d n)$ per update for $d \leq 4$ and in time $O(n^{1-2/(\lceil d/2\rceil+1)+\varepsilon})$ for $d > 4$.*

We note that for rectilinear (L_1 and L_∞) metrics, orthogonal range query data structures can be used to answer dynamic bi-chromatic closest pair queries in $O(\log^d n)$ time per update [34].

Theorem 4.4 *The rectilinear MST of a set of n points in $\mathbb{R}^d$ can be maintained in time $O(\sqrt{n}\log^d n)$ per update.*

References

[1] P.K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Proc. 6th ACM Symp. Comput. Geom.* (1990) 203–210. *SIAM J. Comput.*, to appear.

[2] P.K. Agarwal and J. Matoušek. Ray shooting and parametric search. *Proc. 24th ACM Symp. Theory of Computing* (1992) 517–526.

[3] P.K. Agarwal and M. Sharir. Planar geometric location problems and maintaining the width of a planar set. *Proc. 2nd ACM/SIAM Symp. Discrete Algorithms* (1991) 449–458.

[4] P.K. Agarwal and M. Sharir. Applications of a new partitioning scheme. *Discrete Comput. Geom.* to appear.

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

[6] B. Chazelle, An optimal algorithm for computing convex layers, *IEEE Trans. Information Theory* IT–31 (1985) 509–517.

[7] B. Chazelle, L. Guibas, H. Edelsbrunner, and M. Sharir. Diameter, width, closest line pair, and parametric searching. *Proc. 8th ACM Symp. Computational Geometry* (1992) 120–129.

[8] B. Chazelle, L. Guibas, and D. T. Lee. The power of geometric duality. *BIT* 25 (1985) 76–90.

[9] B. Chazelle and F. P. Preparata. Halfspace range searching: An algorithmic application of k-sets. *Discrete Comput. Geom.* 1 (1986) 83–93.

[10] B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Proc. 6th ACM Symp. Computational Geometry* (1990) 23–33.

[11] Yi Chiang and R. Tamassia. Dynamic algorithms in computational geometry. Tech. Rept. CS-91-24, Dept. Computer Science, Brown University, 1991.

[12] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.* 2 (1987) 195–222.

[13] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.* 17 (1988) 830–847.

[14] K. L. Clarkson and P. Shor. New applications of random sampling in computational geometry II. *Discrete Comput. Geom.* 4 (1989) 387–421.

[15] T. Coreman, C. Leiserson and R. Rivest. *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 1990.

[16] D. Dobkin and S. Suri. Dynamically computing the maxima of decomposable functions, with applications. *J. ACM* 38 (1991) 275–298.

[17] H. Edelsbrunner and M. Overmars. Batched dynamic solutions to decomposable searching problems. *J. Algorithms* 6 (1985) 515–542.

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

[19] D. Eppstein. Dynamic three-dimensional linear programming. *Proc. 32nd IEEE Symp. Found. Computer Science* (1991) 488–494. *ORSA J. Comput.*, to appear.

[20] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. Manuscript, 1992.

[21] D. Eppstein, Z. Galil, G.F. Italiano, and A. Nissentzweig. Sparsification — A technique for speeding up dynamic graph algorithms. *Proc. 32nd IEEE Symp. Found. Computer Science* (1992), this proceedings.

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

[23] J. Hershberger and S. Suri. Offline maintenance of planar configurations. *Proc. 2nd ACM/SIAM Symp. Discrete Algorithms* (1991) 32–41.

[24] J. Matoušek. Reporting points in halfspaces. *Proc. 32nd IEEE Symp. Found. Computer Science* (1991) 207–215.

[25] J. Matoušek and O. Schwarzkopf. Linear optimization queries. *Proc. 8th ACM Symp. Computational Geometry* (1992) 16–25.

[26] N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM J. Computing* 12 (1983) 720–732.

[27] K. Mehlhorn. *Multi-dimensional Searching and Computational Geometry.* Springer-Verlag, Belin–Heidelberg–New York, 1985.

[28] M. Overmars and H. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Sys. Sci.* 23 (1981), 166–204.

[29] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[30] F. Preparata and R. Tamassia. Efficient point location in a spatial cell complex. *SIAM J. Comput.* 21 (1992) 267–280.

[31] N. Sarnak and R. Tarjan. Planar point location using persistent search trees, *Commun. ACM* 29 (1986) 669–679.

[32] M. Smid. Maintaining the minimal distance of a point set in polylogarithmic time. *Proc. 2nd ACM/SIAM Symp. Discrete Algorithms* (1991) 1–6.

[33] K.J. Supowit. New techniques for some dynamic closest-point and farthest-point problems. *Proc. 1st ACM/SIAM Symp. Discrete Algorithms* (1990) 84–90.

[34] P.M. Vaidya. Geometry helps in matching. *SIAM J. Comput.* 18 (1989) 1201–1225.