Algorithmica (1995) 13: 325–345

Algorithmica

Dynamic Half-Space Range Reporting and Its Applications[1]

P. K. Agarwal[2] and J. Matoušek[3]

Abstract. We consider the half-space range-reporting problem: Given a set S of n points in $\mathbb{R}^d$, preprocess it into a data structure, so that, given a query half-space γ, all k points of $S \cap \gamma$ can be reported efficiently. We extend previously known static solutions to dynamic ones, supporting insertions and deletions of points of S. For a given parameter m, $n \le m \le n^{\lfloor d/2 \rfloor}$ and an arbitrarily small positive constant ε, we achieve $O(m^{1+\varepsilon})$ space and preprocessing time, $O((n/m^{\lfloor d/2 \rfloor}) \log n + k)$ query time, and $O(m^{1+\varepsilon}/n)$ amortized update time ($d \ge 3$). We present, among others, the following applications: an $O(n^{1+\varepsilon})$-time algorithm for computing convex layers in $\mathbb{R}^3$, and an output sensitive algorithm for computing a level in an arrangements of planes in $\mathbb{R}^3$, whose time complexity is $O((b+n)\cdot n^{\varepsilon})$, where b is the size of the level.

Key Words. Arrangements, Cuttings, Range-searching, Dynamization.

1. Introduction and Summary of Applications.

We consider the *half-space range-reporting* problem: Given a set S of n points in $\mathbb{R}^d$ (d is a small constant), preprocess it so that, given a query half-space γ, the points of $S \cap \gamma$ can be reported efficiently.

For $d = 2$, a half-plane query can be answered in time $O(\log n + k)$ using $O(n)$ space and $O(n \log n)$ preprocessing, where k is the number of points reported [10]. For $d = 3$, Aggarwal *et al.* [4] have presented an $O(n \log n)$-size data structure that can answer a half-space range query in time $O(\log n + k)$ (see also [11]). For $d > 3$, Clarkson [14] gave a (randomized) solution with $O(n^{\lfloor d/2 \rfloor + \varepsilon})$ space[4] and expected preprocessing time and $O(\log n + k)$ query time. The preprocessing can be made deterministic using the results of [24]. Recently, a solution with $O(n \log n)$ preprocessing time, $O(n \log \log n)$ space, and $O(n^{1-1/\lfloor d/2 \rfloor}(\log n)^{O(1)} + k)$ query time was given in [25]. Combining these last two solutions, we can obtain a space/query time tradeoff: given a parameter m, $n \le m \le n^{\lfloor d/2 \rfloor}$, we can use $O(m^{1+\varepsilon})$ space and preprocessing to achieve $O((n/m^{1/\lfloor d/2 \rfloor}) \log n + k)$ query time. See also [8], [12], [22] for recent works on a related but more general simplex range searching problem.

[1] Work by the first author has been supported by National Science Foundation Grant CCR-91-06514. A preliminary version of this paper appeared in Agarwal *et al.* [2], which also contains the results of [20] on dynamic bichromatic closest pair and minimum spanning trees.

[2] Department of Computer Science, Duke University, Durham, Box 90129, NC 27708-0129, USA.

[3] Department of Applied Mathematics, Charles University, Malostranskénam 25, 118 00 Praha 1, Czech Republic.

[4] Throughout this paper, ε stands for a positive constant which can be chosen arbitrarily small with an appropriate choice of other constants in the algorithms.

Received February 19, 1992; revised December 30, 1992. Communicated by B. Chazelle.

A special case of the half-space range-reporting problems is the *half-space emptiness* problem, where we only want to decide whether the query half-space contains any point of S. For this case, Clarkson's solution [14] yields $O(\log n)$ query time with $O(n^{\lfloor d/2 \rfloor + \varepsilon})$ space and preprocessing time, and Matoušek's solution [25] gives a query time of $O(n^{1-1/\lfloor d/2 \rfloor} \log^{O(1)} n)$ with $O(n)$ space and $O(n \log n)$ preprocessing time. The query time of the latter algorithm can be improved to $O(n^{1-1/\lfloor d/2 \rfloor} 2^{O(\log^* n)})$ if we allow $O(n^{1+\varepsilon})$ preprocessing time. Numerous applications of the half-space range-reporting problem in other computational geometry problems are known and, interestingly, most of them only use the half-space emptiness variant.

All the above-mentioned data structures are static structures, that is, the set S is fixed once for all. In this paper we investigate the situation where it is necessary to modify S dynamically, which is required in many applications. Since the half-space range-reporting problem is decomposable, insertions can be handled using standard dynamization techniques (see [7] and [27]), without affecting the asymptotic running time considerably. The problem is thus how to delete points from the set. For $d = 2$, we can support deletions using the dynamic convex hull maintenance structure of Overmars and van Leeuwen [32]. For higher dimensions, Mulmuley [29] recently presented an $O(n^{\lfloor d/2 \rfloor + \varepsilon})$-size data structure, that for a *random* sequence of insertions and deletions, answers an empty half-space query in $O(\log n)$ time with high probability and performs an update operation in $O(n^{\lfloor d/2 \rfloor 1 + \varepsilon})$ expected amortized time. (However, for some specific update sequences the performance can be bad.)

In this paper we give dynamic counterparts of the half-space range-reporting structures of [14] and [25]. If we sacrifice something in the query answering efficiency, the nearly linear space solution of [25] can be dynamized easily. However, dynamizing Clarkson's structure turns out to be more complicated; we modify it, using a technique due to Chazelle *et al.* [12]. We obtain the following results:

THEOREM 1.1. *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, $O(n^{1-1/\lfloor d/2 \rfloor + \varepsilon} + k)$ query time, and $O(\log^2 n)$ amortized update time.*[5]

(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.*

(iii) *For a parameter m, $n \leq m \leq n^{\lfloor d/2 \rfloor}$, $O(m^{1+\varepsilon})$ space and preprocessing time,*

$$O\left(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log m\right)$$

query time, and $O(m^{1+\varepsilon}/n)$ amortized update time.

[5] By saying that a data structure for a set S of n points has amortized update time $f(n)$, we mean that, starting with a current structure storing S, an arbitrary sequence of at most n insertions and deletions can be performed in $nf(n)$ time. We believe that with some more effort, the amortized bounds could be turned into worst-case ones (for a single update operation), using the standard techniques. However, we feel that this goal is not worth the effort at this point.

Next, we discuss several applications of these data structures to various problems (not trying to provide an exhaustive list). Most of the problems are just stated, since more detailed presentations are given in [3] and [23]. For some problems, solutions are described in the second half of this paper.

Ray Shooting. Given a convex polyhedron P in $\mathbb{R}^d$ defined as the intersection of n half-spaces, preprocess it into a data structure, so that the first intersection point of the boundary of P and a query ray can be computed quickly. In another paper [3], we describe a data structure for this problem with $O(m^{1+\varepsilon})$ space $(n \leq m \leq n^{\lfloor d/2 \rfloor})$, $O((n/m^{1/\lfloor d/2 \rfloor}) \log^5 n)$ query time, and $O(m^{1+\varepsilon}/n)$ amortized update time (for inserting or deleting a half-space determining P). The dynamic solution uses our dynamic half-space emptiness data structure. If the ray originates inside P, the query time can be improved by a $\log^3 n$ factor. Further minor improvements in the query time are discussed in [23]. As we will see later, several other problems can also be reduced to answering ray-shooting queries in a convex polyhedron.

Half-space range reporting has also been applied to some other ray-shooting problems, see [3].

Nearest/Farthest-Neighbor Queries. Given a set S of n points in $\mathbb{R}^d$, preprocess it into a data structure, so that the k nearest (or k farthest) neighbors of a query point can be determined quickly (k can be a part of the query). The static version of the problem is well studied [4]. Since the problem is decomposable, it can be easily dynamized efficiently if we allow only insertions, and if we are willing to pay an extra logarithmic factor in the query time.

The problem however becomes much harder if we allow deletions. For $d = 2$, the query and the update time for the best-known nearest- (or farthest-) neighbor algorithm are $O(\sqrt{n} \log n)$. (Of course, if we allow linear update time, a query can be answered in $O(\log n)$ time.) We are not aware of any efficient solution in higher dimensions except in some special cases, e.g., the closest pair in a set of n points can be maintained in $O(\log^d n \log \log n)$ amortized time [34] (see also [35]). Recently Mulmuley [29] showed that if the sequence of insertions/deletions is random, then a k-nearest-neighbor query can be answered in $O(k \log n)$ expected time, and a point can be inserted or deleted in amortized time $O(n^{\lceil d/2 \rceil - 1 + \varepsilon})$ with high probability (other variants of the algorithm give slightly better expected performance bounds, but not with high probability, see [29]). Again, the algorithm is not guaranteed to perform very well on an arbitrary sequence of insertions and deletions.

In [3] we gave a data structure of size $O(m^{1+\varepsilon})$ that could answer a k-nearest- (or k-farthest-) neighbor query in time $O((n/m^{1/\lceil d/2 \rceil}) \log^{O(1)} n + k \log^2 n)$. Using the dynamic half-space range reporting structure, we get, as usual, amortized update time $O(m^{1+\varepsilon}/n)$ for a dynamic version of this data structure. The same bounds hold if we want to determine the k farthest neighbors. Let us remark that if $m = n^{\lceil d/2 \rceil}$, the query time can be improved to $O(\log^2 n + k \log n)$, which matches Mulmuley's bound for $k = \Omega(\log n)$.

Dynamic Linear Optimization Queries. Let H be a set of n linear constraints (half-spaces) in $\mathbb{R}^d$. We wish to preprocess H, so that the optimal point with respect to a query objective function (hyperplane) can be computed efficiently.

For $d = 2$, the dynamic version of the above problem can be solved, efficiently using the Overmars and van Leeuwen algorithm for maintaining convex hulls [32]. For $d = 3$, Eppstein [19] recently gave a data structure for the special case in which the sequence of insertions/deletions is known in advance. For $d \leq 4$, assuming a random sequence of updates, Mulmuley [30] gave an algorithm that answers a query in polylogarithmic time with high probability, and updates the set of hyperplanes in expected amortized time $O(\log n)$ and $O(n \log^2 n)$ for $d = 3$, 4, respectively.

It was shown in [23] that a data structure for the half-space emptiness problem (satisfying certain mild assumptions, met by our data structure) can be used for answering linear optimization queries, with a polylogarithmic number of half-space emptiness queries needed for answering an optimization query. Hence, given a parameter m, $n \leq m \leq n^{\lceil d/2 \rfloor}$, a data structure can be maintained for a set of n linear constraints in $\mathbb{R}^d$, with $O(m^{1+\varepsilon})$ space and $O(m^{1+\varepsilon}/n)$ amortized update time, so that the optimal solution for a query objective function can be computed in time $O((n/m^{\lfloor d/2 \rfloor}) \log^{O(1)} n)$. The same algorithm can be used to maintain the convex hull of a set of points implicitly, so that various queries, e.g., whether a hyperplane intersects the convex hull of S, or whether a query point lies in the convex hull of S, etc., can be answered efficiently.

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

Maintaining Bichromatic Closest Pair and Diameter. Given a set of red points and another set of blue points in $\mathbb{R}^d$, maintain a closest red–blue pair of points. Agarwal *et al.* [1] gave an $O(n^{2-2/(\lceil d/2 \rceil + 1)+\varepsilon})$-time algorithm to compute a bichromatic closest pair of a set of n points in $\mathbb{R}^d$. However, no efficient dynamic algorithm was known for this problem. Recently Eppstein [20] has shown that a dynamic data structure for nearest-neighbor searching can be used to maintain a bichromatic closest pair efficiently. Plugging our data structure into his algorithm, a bichromatic closest pair in $\mathbb{R}^d$ can be maintained in time $O(n^{1-2/(\lceil d/2 \rceil + 1)+\varepsilon})$. It also yields a dynamic algorithm for maintaining a Euclidean minimum spanning tree of n points in the plane with $O(\sqrt{n} \log n)$ update time.

The diameter of a set S of points is the distance between the farthest pair of points in S. It is well known that the diameter of a set of n points in the plane can be computed in time $O(n \log n)$. The algorithm of Agarwal *et al.* [1] can be modified to compute the diameter as well. Recently, Chazelle *et al.* [9] presented an $O(n^{1+\varepsilon})$-time algorithm to compute the diameter of a set of points in $\mathbb{R}^3$. Supowit [36] presented a data structure that could update the diameter of a set of n points in the plane in $O(\sqrt{n} \log n)$ amortized time if only deletions were allowed. However, no efficient algorithm was known if both insertions and deletions are allowed. The algorithm of Eppstein [20] can be modified for maintaining the diameter of a set of points. In particular, the diameter of a set of n points in $\mathbb{R}^d$ can be maintained in time $O(n^{1-2/(\lceil d/2 \rceil + 1)+\varepsilon})$.

Convex Layers in Dimension 3. For a finite set $S \subset \mathbb{R}^d$, its convex layers C_1,

$C_2, \ldots$ are defined inductively as follows: Let $CH(S)$ denote the set of points lying on the boundary of the convex hull of S. Let $S_0 = S$ and, for $i \geq 1$, $C_i = CH(S_{i-1})$ and $S_i = S_{i-1} - (C_i \cap S_{i-1})$. In Section 3 we present an $O(n^{1+\varepsilon})$-time algorithm for computing the convex layers of a set of n points in $\mathbb{R}^3$; the previously best-known algorithm required $O(n^{3/2} \log n)$ time.

Levels in Arrangements, Higher-Order Voronoi Diagrams. Let H be a set of n hyperplanes in $\mathbb{R}^d$. The *level* of a facet f in $\mathcal{A}(H)$ is the number of hyperplanes of H lying strictly above f. The *k-level* $\Pi_k = \Pi_k(H)$ in the arrangement of H is the set of (the closures of) facets whose level is k. Let $|\Pi_k|$ denote the total number of faces of all dimensions in Π_k (the complexity of Π_k). In Section 4 we present an output-sensitive algorithm for computing a level in dimension 3, with time complexity $O((b + n)n^{\varepsilon})$, where b is the complexity of the level.

As a corollary, we can compute the kth-order Voronoi diagram of n points in the plane in time $O(n^{1+\varepsilon} k)$.

Our algorithm for computing a level can be extended to higher dimensions, though the running time becomes worse.

2. Dynamizing Half-Space Range Reporting. In this section we prove Theorem 1.1. We begin by observing that 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 is known that a data structure for a decomposable problem, which supports delete operations, can be converted into another data structure that supports insertions as well [27]. In particular,

THEOREM 2.1. *Given a set S of n points, let $\Psi(S)$ be a data structure for the half-space range-reporting problem that supports deletions. Let $P(n)$ be the preprocessing time, $Q(n) + O(k)$ the query time, and $D(n)$ the amortized deletion time of $\Psi(S)$. Let β be a parameter, not necessarily constant. Then another data structure can be constructed that answers a half-space range-reporting query in time $O(k) + \sum_{i=1}^{\lceil \log_\beta n \rceil} Q(\beta^i)$, can insert a point in amortized time $O(\sum_{i=0}^{\lceil \log_\beta n \rceil} P(\beta^{i+1})/\beta^i)$, and can delete a point in amortized time $D(n) + O(\log n + P(n)/n)$. The size and preprocessing time of the new data structure remain the same.*

A proof of a similar claim is given in [27] (see also [31]), but for the sake of completeness we present the proof here.

PROOF. Let S be the set of input points in $\mathbb{R}^d$. The current set S is partitioned into $t = \lceil \log_\beta n \rceil$ subsets $S_1, \ldots, S_t$, such that S_i contains at most $(\beta - 1)\beta^i$ points. For each $i \leq t$, we maintain $\Psi(S_i)$ and a counter r_i (varying in range from 0 to $\beta - 1$). Let γ be a query half-space. To report the points of $S \cap \gamma$, we query all $\Psi(S_i)$ with γ and form the union of the (pairwise disjoint) answers.

Next consider the deletion of a point from S. As we delete points from S, we periodically reconstruct the entire structure. We remember the number of points deleted from S since it was constructed the last time, and we always reconstruct

$\Psi(S)$ anew after deleting $n'/2$ points, where n' is the number of points in S when the structure was constructed the last time (the entire structure may be reconstructed while inserting a point too; see below). When reconstructing the data structure with an n-point current set S, we let $t = \lceil \log_\beta n \rceil$. We set $S_t = S$ and $S_i = \emptyset$ for $i < t$, and construct $\Psi(S_t)$. We set the counter $r_i = 0$ for $i < t$ and $r_t = \lceil |S_t|/\beta^t \rceil$. The time spent in periodic reconstruction of the structure can be subsumed by charging an additional $O(P(n)/n)$ time to each insert/delete operation.

While deleting a point p, if the data structure is not reconstructed, we proceed as follows. We find in $O(\log n)$ time the set S_i that contains p, delete p from S_i, and then modify $\Psi(S_i)$ accordingly. The total amortized running time of a delete operation is thus at most $D(n) + O(\log n + P(n)/n)$.

We insert a point to S using the following procedure:

Algorithm INSERT (S, p)
$P \leftarrow \{p\}$, $i \leftarrow 0$
while $r_i = \beta - 1$ **do**
  $P \leftarrow P \cup S_i$
  $S_i \leftarrow \emptyset$, $r_i \leftarrow 0$
  $i \leftarrow i + 1$
end while
$S_i \leftarrow P$, $r_i \leftarrow r_i + 1$
Construct $\Psi(S_i)$

It is easily seen that $|S_i| < \beta^{i+1}$. Since we spend at most $P(\beta^{i+1})$ time in constructing S_i and each S_i is constructed only after performing n/β^i insert operations, the claim follows. □

In view of the above theorem, it suffices to describe half-space reporting data structures that support only delete operations.

2.1. The Case of Almost Linear Space. We now describe how to dynamize the data structure of [25]. The query time of our data structure is slightly worse than that of [25], but our data structure is somewhat simpler. For simplicity, let us assume that the points in S are in general position. We need some definitions.

DEFINITION 2.2. Let S be a set of n points. A hyperplane h is called *k-shallow* (with respect to S), if one of the open half-spaces determined by h contains at most k points. A *simplicial partition* for S is a collection $\Pi = \{(S_1, \Delta_1), (S_2, \Delta_2), \ldots, (S_t, \Delta_t)\}$, where $S_1, \ldots, S_t$ form a disjoint partition of S and each Δ_i is a simplex containing S_i.

We use the following result:

THEOREM 2.3 [25]. *Let S be a set of n points in $\mathbb{R}^d$ $(d \geq 3)$, and let k, r be parameters such that $k \leq n/r$, $1 \leq r < n$. Then a simplicial partition $\Pi = \{(S_1, \Delta_1), \ldots, (S_t, \Delta_t)\}$*