Output-Sensitive Results on Convex Hulls, Extreme Points, and Related Problems

Timothy M. Chan*
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1Z4, Canada

Abstract

We use known data structures for ray shooting and linear programming queries to derive new output-sensitive results on convex hulls, extreme points, and related problems. We show that the f-face convex hull of an n-point set P in a fixed dimension $d \geq 2$ can be constructed in $O(n \log f + (nf)^{1-1/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n)$ time; this is optimal if $f = O(n^{1/\lfloor d/2 \rfloor} / \log^K n)$ for some sufficiently large constant K. We also show that the h extreme points of P can be computed in $O(n \log^{O(1)} h + (nh)^{1-1/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n)$ time. These results are then applied to produce an algorithm that computes the vertices of all the convex layers of P in $O(n^{2-\gamma})$ time for any constant $\gamma < 2/(\lfloor d/2 \rfloor^2 + 1)$. Finally, we obtain improved time bounds for other problems including levels in arrangements and linear programming with few violated constraints. In all of our algorithms, the input is assumed to be in general position.

1 Introduction

Let P be a set of n points in d-dimensional Euclidean space E^d, where $d \geq 2$ is a fixed constant. We assume that the points are in general position. The smallest convex set containing P is a polytope conv(P) called the *convex hull* of P. It is known that the number of faces, f, in this polytope is at worst $\Theta(n^{\lfloor d/2 \rfloor})$ [27]. In the convex hull problem, we want to construct the facial structure of conv(P). This problem has been intensively studied in computational geometry [16, 31, 33, 36], and it has applications to other geometric problems such as computing intersections of halfspaces and computing Voronoi diagrams and Delaunay triangulations.

Chazelle [10] has solved the convex hull problem optimally in the worst case by giving an $O(n \log n + n^{\lfloor d/2 \rfloor})$-time algorithm. However, this bound depends only on the input size n and is insensitive to the output size f. An optimal $O(n \log f)$-time output-sensitive algorithm in two dimensions was given by Kirkpatrick and Seidel [23]. For dimension 3, Edelsbrunner and Shi [17] obtained an $O(n \log^2 f)$-time method, and Chazelle and Matoušek [11] demonstrated that optimal $O(n \log f)$ time is possible by derandomizing an earlier algorithm due to Clarkson and Shor [13]. In any fixed dimension, the "gift-wrapping" algorithm of Swart [44] and the "beneath/beyond" algorithm of Seidel [41] achieve $O(nf)$ and $O(n^2 + f \log n)$ time respectively. The latter is subsequently improved

*Supported by a Killam Predoctoral Fellowship and an NSERC Postgraduate Scholarship.

to $O(n^{2-2/(\lfloor d/2\rfloor+1)+\varepsilon}+f\log n)$ by Matoušek [24] using a data structuring technique that he has developed for linear programming queries. (Throughout this paper, $\varepsilon>0$ denotes an arbitrarily small constant.) It is in fact possible to reduce the $O(n^{2-2/(\lfloor d/2\rfloor+1)+\varepsilon})$ term to $O(n^{2-2/(\lfloor d/2\rfloor+1)}\log^{O(1)}n)$ by using the static structures in [24]. Recently, Chan, Snoeyink, and Yap [7] have obtained an $O((n+f)\log^2 f)$-time algorithm in four dimensions; in higher dimensions, their method is less efficient, running in $O((n+f^{d-3})\log^{d-2}f)$ time. Thus, there is still a large gap between the known upper bounds and the $\Omega(n\log f+f)$ lower bound for $d>4$.

Here, we show that the gift-wrapping method can be further improved using the data structures for ray shooting queries in polytopes developed by Agarwal and Matoušek [1] and refined by Matoušek and Schwarzkopf [26]. Our convex hull algorithm runs in $O(n\log f+(nf)^{1-1/(\lfloor d/2\rfloor+1)}\log^{O(1)}n)$ time and is optimal when $f=O(n^{1/\lfloor d/2\rfloor}/\log^K n)$ for a sufficiently large K. Furthermore, it is faster than all previous methods when $f=O(n/\log^K n)$ and $d>4$. Note that in many cases, f can in fact be sublinear; for example, Raynaud [37] proved that the expected value of f is $O(n^{(d-1)/(d+1)})$ if the points of P are chosen uniformly at random from a d-dimensional ball. The expected number of hull vertices is only polylogarithmic in n if the points are chosen uniformly from a hypercube or from a normal distribution [3, 37].

Surprisingly, our method leads to new optimal output-sensitive algorithms in two and three dimensions, running in $O(n\log f)$ time. In the plane, our algorithm is as simple as Kirkpatrick and Seidel's, and in three dimensions, our algorithm is simpler than Chazelle and Matoušek's. These are reported separately [6].

Next, we turn to the problem of computing the *extreme points* of P, i.e., the vertices of $\mathrm{conv}(P)$ (or equivalently, the set of points $p\in P$ with $\mathrm{conv}(P-\{p\})\neq\mathrm{conv}(P)$). By Megiddo's linear programming algorithm [29], we can test whether a given point is an extreme point of P in linear time; this immediately yields an algorithm for the extreme point problem that runs in $O(n^2)$ time. Matoušek reduced the bound to $O(n^{2-2/(\lfloor d/2\rfloor+1)+\varepsilon})$ using his data structures for linear programming queries [24]; again, the n^ε factor can be replaced by $\log^{O(1)}n$ if static structures are used. We further improve this to an output-sensitive bound. Let h denote the number of extreme points ($h\leq n$). By using Matoušek's data structures in a simple $O(nh)$-time algorithm, we show that the extreme points can be computed in $O(n\log^{O(1)}h+(nh)^{1-1/(\lfloor d/2\rfloor+1)}\log^{O(1)}n)$ time. This $O(nh)$-time algorithm has recently been discovered independently by Clarkson [12] and Ottmann et al. [32].

We then consider the problem of computing the *convex layers* of P, defined iteratively as follows: layer 1 is the convex hull of P, and if layer i is nonempty, then layer $i+1$ is defined as the convex hull of the points of P that are not vertices of the previous layers $1,\ldots,i$. It is known that this problem can be solved optimally in $O(n\log n)$ time by an algorithm of Chazelle [9] for $d=2$ and quasi-optimally in $O(n^{1+\varepsilon})$ time by an algorithm of Agarwal and Matoušek [2] for $d=3$.

For $d\geq 4$, Edelsbrunner [16, Problem 10.3(c)] asked whether the vertices of all layers can be identified in $o(n^3)$ time. This problem is equivalent to finding the *depth* of p, i.e., the index of the layer of which p is a vertex, for every $p\in P$. It is not difficult to get an $O(n^{3-3/(\lfloor d/2\rfloor+1)+\varepsilon})$-time solution by applying Matoušek's technique [24]. As Ottmann et al. [32] have pointed out, an $O(nh)$-time extreme point algorithm [12] immediately yields an $O(n^2)$-time solution. Here, we show how the depth problem can be solved in $O(n^{2-\beta+\varepsilon})$ time with $\beta=2/(\lfloor d/2\rfloor^2+1)$; for example, in four or five dimensions, the bound is $O(n^{8/5+\varepsilon})$. As a result, we can construct the convex layers in

$O(n^{2-\beta+\varepsilon} + f \log n)$ time, where f is now the total number of faces in all layers (which is at least $\Omega(n)$ and at most $O(n^{\lfloor d/2 \rfloor})$).

Finally, we examine applications of our ideas to other related problems. The first application we consider is the construction of a level in an arrangement of hyperplanes. Given a set H of n hyperplanes in E^d, the *k-level* in the arrangement $\mathcal{A}(H)$ is defined as the set of all points in E^d that have at most k hyperplanes of H above it ($0 \leq k < n$). The 0-level is just the dual of a convex hull. In the plane, an output-sensitive algorithm for constructing the k-level was given by Edelsbrunner and Welzl [19]. We improve its running time from $O(n \log n + f \log^2 n)$ to $O(n \log f + f \log^2 n)$, where f denotes the size of the k-level. In higher dimensions, Agarwal and Matoušek [2] proposed a method based on ray shooting queries, which runs in $O(n \log n + f^{1+\varepsilon})$ time for $d = 3$ (actually they state a weaker $O((n + f)n^{\varepsilon})$ bound). We improve this to $O(n \log f + f^{1+\varepsilon})$ and show that the time bound is $O(n \log f + (nf)^{1-1/(\lfloor d/2 \rfloor+1)+\varepsilon} + fn^{1-2/(\lfloor d/2 \rfloor+1)+\varepsilon})$ for $d \geq 4$.

Another related problem studied here is: given a set H of n hyperplanes in E^d, a direction ξ, and a small integer $0 \leq k < n$, find a point in the k-level of $\mathcal{A}(H)$ that is minimal w.r.t. ξ; in other words, find a minimal point that lies on or above all but at most k of the hyperplanes in H. This is the feasible case of the *linear programming problem with at most k violated constraints*. For this problem, Matoušek [25] has devised a method that not only finds the minimum but also enumerates all $O(k^d)$ local minima in the ($\leq k$)-levels. His method runs in $O(n \log n + k^2 \log^2 n)$ time if $d = 2$ and $O(n \log n + k^{3+\varepsilon})$ time if $d = 3$; when $d \geq 4$ and k is sufficiently small (more precisely: $k^d \leq n^{1/\lfloor d/2 \rfloor+\varepsilon}$), the running time is $O(n \log n)$. We show how the $O(n \log n)$ terms in these bounds can be reduced to $O(n \log k)$ in two dimensions or to $O(n \log \log n + n \log k)$ in higher dimensions; if randomization is allowed, this $O(n \log \log n)$ term can even be removed.

As an aside, we point out that the Matoušek's results [25] can be used to improve an algorithm by Mulmuley [30] for constructing ($\leq k$)-levels of a nonredundant arrangement of n hyperplanes in E^d. The algorithm is an extension of Seidel's output-sensitive convex hull algorithm [41] and runs in $O(n^2 k^{d-1} + f \log n)$ time for an f-face output. We decrease the time bound to $O(n^{2-2/(\lfloor d/2 \rfloor+1)+\varepsilon} k^{d-1} + f \log n)$.

We remark that all our algorithms depend heavily on the assumption that the input points or hyperplanes are in general position. For some of the problems we have considered (e.g., convex hulls and k-levels), standard perturbation techniques [18, 20] may be used when this assumption does not hold. However, one should keep in mind that these perturbation methods may increase the output size when there is a large number of degeneracies.

The remainder of this paper is organized as follows. In Sections 2 and 3, we review some of the known data structures for ray shooting queries and linear programming queries, which serve as the basic tools of our approach. Our contribution is a (very simple) preprocessing time/query time tradeoff that allows us to obtain improved time bounds when there are only a small number of queries. We then apply these results to the output-sensitive construction of convex hulls in Section 4 and the output-sensitive computation of extreme points in Section 5. Applications to convex layers and depths are discussed in Section 6; further applications are given in Section 7. We then conclude with some final remarks in Section 8.

2 Ray Shooting Queries

We first investigate the problem of ray shooting in polytopes. Let H be a collection of n (closed) halfspaces in E^d, where each halfspace contains a known point, say, the origin o. Let $\bigcap H$ denote the intersection of these halfspaces. Without loss of generality, we assume that $\bigcap H$ is bounded (otherwise, we intersect it with a sufficiently large box and consider this intersection instead). Then $\bigcap H$ is a (convex) polytope. A *ray shooting query* is to determine the first bounding hyperplane h of $\bigcap H$ that is crossed by a query ray originating from some point in $\bigcap H$ (a ray *crosses* a hyperplane h if it intersects h but is not contained in h).

In two dimensions, the ray shooting problem can be solved as follows: first compute the polygon $\bigcap H$ and store its vertices in an array in counterclockwise order; then a query can be done by a simple binary search. Observe that computing the intersection $\bigcap H$ is equivalent to computing a convex hull in the dual space, and thus takes $O(n \log n)$ time by Graham's scan for example [22]; and the binary search takes $O(\log n)$ time. Hence, this method requires $O(n \log n)$ preprocessing time, $O(n)$ space, and $O(\log n)$ query time.

The same preprocessing time, space, and query time can be obtained in three dimensions: in the preprocessing, compute the polytope $\bigcap H$ by the dual of Preparata and Hong's convex hull algorithm [35] and construct its Dobkin-Kirkpatrick hierarchical representation [15]; then use the query algorithm from [15].

Our first observation is that a preprocessing time/query time tradeoff is possible using a standard "grouping" technique. Using this observation, we can perform q queries in $O(n \log q)$ time rather than $O(n \log n)$ time for small q's.

Lemma 2.1 *There is a (static) data structure for ray shooting in a polytope defined by a set H of n halfspaces in E^2 or E^3 with $O(n \log m)$ preprocessing time, $O(n)$ space, and $O((n/m) \log m)$ query time, where m is a parameter between 1 and n.*

Proof: Partition H into $\lceil n/m \rceil$ subsets ("groups") $H_1, \ldots, H_{\lceil n/m \rceil}$, each of size at most m and build the above structures for each H_i. The total preprocessing time is $O(\frac{n}{m}(m \log m)) = O(n \log m)$, and the space complexity remains $O(n)$. Since ray shooting is a *decomposable* problem (i.e., the answer to a query on $H' \cup H''$ can be computed from the answers to the queries on H' and H'' in constant time), a query on H can be computed directly by querying on each H_i, taking $O((n/m) \log m)$ time. $\square$

Corollary 2.2 *An (online) sequence of q ray shooting queries in a polytope defined by a set H of n halfspaces in E^2 or E^3 can be performed in $O(n \log q + q \log n)$ time and $O(n)$ space.*

Proof: By Lemma 2.1, the total time needed to answer q queries is $O(n \log m + q(\frac{n}{m} \log m))$, where $1 \leq m \leq n$. Choose $m = q$ when $q \leq n$ and choose $m = n$ when $q > n$. $\square$

For ray shooting queries in d-dimensional polytopes, Agarwal and Matoušek [1] have proposed a data structuring method that was subsequently improved by Matoušek and Schwarzkopf [26]. The top half of Table 1 shows their results in the static case. (The table also includes results on linear programming queries, which are discussed in the next section.) Structure 1 is a linear-space data structure with $O(n \log n)$ preprocessing time and $O(n^{1-1/\lfloor d/2 \rfloor} \log^{O(1)} n)$ query time. Structure 3

Structures	preprocessing time, space	update time (amortized)	ray shooting query time	linear programming query time
1	$n \log n,\ \ n$	N/A	$n^{1-1/\lfloor d/2\rfloor} \log^{O(1)} n$	$n^{1-1/\lfloor d/2\rfloor} \log^{O(1)} n$
2	$m \log^{O(1)} n$	N/A	$\frac{n}{m^{1/\lfloor d/2\rfloor}} \log n$	$\frac{n}{m^{1/\lfloor d/2\rfloor}} \log^{2d+1} n$
3	$n^{\lfloor d/2\rfloor} \log^{O(1)} n$	N/A	$\log n$	$\log^{d+1} n$
$1'$	$n \log n,\ \ n$	$\log^2 n$	$n^{1-1/\lfloor d/2\rfloor+\varepsilon}$	$n^{1-1/\lfloor d/2\rfloor+\varepsilon}$
$2'$	$m^{1+\varepsilon}$	$m^{1+\varepsilon}/n$	$\frac{n}{m^{1/\lfloor d/2\rfloor}} \log n$	$\frac{n}{m^{1/\lfloor d/2\rfloor}} \log^{2d+1} n$
$3'$	$n^{\lfloor d/2\rfloor+\varepsilon}$	$n^{\lfloor d/2\rfloor-1+\varepsilon}$	$\log n$	$\log^{d+1} n$

Table 1: Known (deterministic) data structures for ray shooting queries in polytopes [26] and linear programming queries [24]. For Structures 2 and $2'$, m is a parameter between n and $n^{\lfloor d/2\rfloor}$.

achieves a much faster query time of $O(\log n)$, but preprocessing time increases to $O(n^{\lfloor d/2\rfloor} \log^{O(1)} n)$. A continuous tradeoff between preprocessing and query time is provided by Structure 2.

We observe here that the grouping technique can be used to obtain further preprocessing time/query time tradeoffs for Structure 1.

Lemma 2.3 *There is a (static) data structure for ray shooting in a polytope defined by a set H of n halfspaces in E^d $(d > 3)$ with $O(n \log m)$ preprocessing time, $O(n)$ space, and $O((n/m^{1/\lfloor d/2\rfloor}) \log^{O(1)} m)$ query time, where m is a parameter between 1 and n.*

Proof: By partitioning H into $\lceil n/m\rceil$ groups as in Lemma 2.1 and using Structure 1 to store each group, the preprocessing time becomes $O(\frac{n}{m}(m \log m)) = O(n \log m)$ and query time becomes $O(\frac{n}{m}(m^{1-1/\lfloor d/2\rfloor} \log^{O(1)} m)) = O(\frac{n}{m^{1/\lfloor d/2\rfloor}} \log^{O(1)} m)$. □

Corollary 2.4 *A sequence of q ray shooting queries in a polytope defined by a set H of n halfspaces in E^d $(d > 3)$ can be performed in $O(n \log q + (nq)^{1-1/(\lfloor d/2\rfloor+1)} \log^{O(1)} n + q \log n)$ time.*

Proof:

Case I. $q < n^{1/\lfloor d/2\rfloor}/\log^K n$, where K is a sufficiently large constant. Use Lemma 2.3's modification of Structure 1 with $m = (q \log^K q)^{\lfloor d/2\rfloor}$ $(1 \le m \le n)$. Then the running time is

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

Case II. $n^{1/\lfloor d/2\rfloor}/\log^K n \le q \le n^{\lfloor d/2\rfloor}/\log^K n$. Use Structure 2 with $m = (nq \log^K n)^{1-1/(\lfloor d/2\rfloor+1)}$ $(n \le m \le n^{\lfloor d/2\rfloor})$. Then the running time is

$$O\left(m \log^{O(1)} n + q\frac{n}{m^{1/\lfloor d/2\rfloor}} \log n\right) = O((nq)^{1-1/(\lfloor d/2\rfloor+1)} \log^{O(1)} n).$$

Case III. $q > n^{\lfloor d/2\rfloor}/\log^K n$. Use Structure 3. Then the running time is

$$O\left(n^{\lfloor d/2\rfloor} \log^{O(1)} n + q \log n\right) = O((nq)^{1-1/(\lfloor d/2\rfloor+1)} \log^{O(1)} n + q \log n).$$

□

Remark: In some applications, the number of queries q may not be known in advance. In that case, the parameter m cannot be set directly. This problem can be avoided by breaking the q queries into k clusters of $q_1, \ldots, q_k$ queries, where $q_1, q_2, \ldots$ is a known sequence and $q_1 + \ldots + q_{k-1} < q \leq q_1 + \ldots + q_k$. For example, in Case I of the proof of Corollary 2.4, if we choose the sequence $q_i = 2^{2^i}$ $(i = 1, 2, \ldots)$, then the total running time is $O(\sum_{i=1}^{k} n \log q_i) = O(\sum_{i=1}^{\lceil \log\log q \rceil} n 2^i) = O(n \log q)$, as before. (Logarithms are in base 2.) As soon as q exceeds $n^{1/\lfloor d/2 \rfloor} / \log^K n$, we switch to Case II and use a different sequence $q_i = n^{1/\lfloor d/2 \rfloor} 2^i$ $(i = 1, 2, \ldots)$; the running time is again a geometric series and thus increases by only a constant factor. Finally, when q exceeds $n^{\lfloor d/2 \rfloor} / \log^K n$, we switch to Case III, which does not require setting the parameter m.

We now discuss dynamic ray shooting in polytopes, where halfspaces may be inserted or deleted. Let n denote a (known) upper bound on the number of halfspaces at any given time. In two dimensions, a data structure by Overmars and van Leeuwen [34] has $O(n \log n)$ preprocessing time, $O(n)$ space, $O(\log^2 n)$ update time, and $O(\log n)$ query time. We can extend the grouping technique to get the following analogues of Lemma 2.1 and Corollary 2.2:

Lemma 2.5 *There is a data structure for ray shooting in a polygon defined by a dynamic set H of at most n halfplanes in E^2 with $O(n \log m)$ preprocessing time, $O(n)$ space, $O((n/m) \log^2 m)$ update time, and $O((n/m) \log m)$ query time $(1 \leq m \leq n)$.*

Proof: Let $B_1, \ldots, B_{\lceil n/m \rceil}$ be $\lceil n/m \rceil$ "buckets", each containing at most m halfplanes at any given time. For each bucket, keep a dictionary of its halfplanes and use Overmars and van Leeuwen's data structure to store the polygon defined by these halfplanes. The preprocessing time, space complexity, and query time are as before. To perform an insertion, we search for a bucket B_i that is not full (i.e., one that contains $< m$ halfplanes) and insert the given halfplane to B_i; to perform a deletion, we search for the bucket B_i containing the given halfplane and delete it from B_i. Clearly, the update time is upper-bounded by $O((n/m) \log^2 m)$. (In fact, insertions can be done in $O(\log^2 m)$ time if we maintain a linked list of buckets that are not full; deletions can also be done in $O(\log^2 m)$ time if we are given a pointer to the halfplane being deleted.) $\square$

Corollary 2.6 *A sequence of q ray shooting queries in a polygon defined by a dynamic set H of at most n halfplanes in E^2, and q insertions/deletions on H can be performed in $O(n \log q + q \log^2 n)$ time and $O(n)$ space.*

Proof: By Lemma 2.5, the total time needed to perform q queries and updates is $O(n \log m + q(\frac{n}{m} \log^2 m))$, where $1 \leq m \leq n$. Choose $m = q \log q$ when $q \leq n/\log n$ and choose $m = n$ otherwise. $\square$

In higher dimensions, Matoušek and Schwarzkopf [26] have provided dynamic versions of their data structures, as shown in the bottom half of Table 1. These dynamic structures, named Structures $1'$, $2'$, and $3'$ in the table, achieve the same preprocessing and query time as their static counterparts, except that certain $\log^{O(1)} n$ factors are increased to n^{ε}. With the techniques we have used so far, it is straightforward to obtain a modification of Structure $1'$ with the following tradeoff: $O(n \log m)$ preprocessing time, $O(n)$ space, $O((n/m) \log^2 m)$ amortized update time, and $O(n/m^{1/\lfloor d/2 \rfloor - \varepsilon})$ query time $(1 \leq m \leq n)$. We then have:

Lemma 2.7 *A sequence of q ray shooting queries in a polytope defined by a dynamic set H of at most n halfspaces in E^d $(d > 2)$ can be performed in*

i. $O(n\log q + (nq)^{1-1/(\lfloor d/2\rfloor+1)+\varepsilon} + qn^{1-2/(\lfloor d/2\rfloor+1)+\varepsilon})$ *time, if the number of insertions/deletions is $O(q)$;*

ii. $O(n\log^2 n + (nq)^{1-1/(\lfloor d/2\rfloor+1)+\varepsilon} + q\log n)$ *time, if the number of insertions/deletions is $O(n)$.*

Proof: For (i), we consider three cases:

CASE I. $q < n^{1/\lfloor d/2\rfloor-\varepsilon}$. Use the above modification of Structure 1′ with $m^{1/\lfloor d/2\rfloor-\varepsilon} = q$ $(1 \le m \le n)$. Then the running time is

$$O\left(n\log m + q\frac{n}{m}\log^2 m + q\frac{n}{m^{1/\lfloor d/2\rfloor-\varepsilon}}\right) = O(n\log q).$$

CASE II. $n^{1/\lfloor d/2\rfloor-\varepsilon} \le q \le n^{1-\varepsilon}$. Use Structure 2′ with $m = (n^{1+\varepsilon}q)^{1-1/(\lfloor d/2\rfloor+1)}$ $(n \le m \le n^{\lfloor d/2\rfloor})$. Then the running time is

$$O\left(m^{1+\varepsilon} + q\frac{m^{1+\varepsilon}}{n} + q\frac{n}{m^{1/\lfloor d/2\rfloor}}\log n\right) = O((nq)^{1-1/(\lfloor d/2\rfloor+1)+\varepsilon}).$$

CASE III. $q > n^{1-\varepsilon}$. Use Structure 2′ with $m = n^{2-2/(\lfloor d/2\rfloor+1)}$ $(n \le m \le n^{\lfloor d/2\rfloor})$. Then the running time is

$$O\left(m^{1+\varepsilon} + q\frac{m^{1+\varepsilon}}{n} + q\frac{n}{m^{1/\lfloor d/2\rfloor}}\log n\right) = O(qn^{1-2/(\lfloor d/2\rfloor+1)+2\varepsilon}).$$

For (ii), we perform a similar analysis. We use Structure 1′ when $q < n^{1/\lfloor d/2\rfloor-\varepsilon}$, Structure 2′ with $m = (n^{1+\varepsilon}q)^{1-1/(\lfloor d/2\rfloor+1)}$ when $n^{1/\lfloor d/2\rfloor-\varepsilon} \le q \le n^{\lfloor d/2\rfloor-\varepsilon}$, and Structure 3′ when $q > n^{\lfloor d/2\rfloor-\varepsilon}$. □

Remark: As in the previous remark, the value of q does not need to be given in advance.

3 Linear Programming Queries

Let H be a collection of halfspaces in E^d as in Section 2. A *linear programming query* is to determine the vertex v of the polytope $\bigcap H$ that maximizes $\xi \cdot v$ for a query vector $\xi \in E^d$.

We begin by extending the grouping technique of Lemmas 2.1 and 2.3 to handle linear programming with a small number of queries. This is not trivial because linear programming, unlike ray shooting, is not a decomposable problem.

Lemma 3.1 *There is a data structure for linear programming queries on a dynamic set H of at most n halfplanes in E^2 with $O(n\log m)$ preprocessing time, $O(n)$ space, and $O((n/m)\log^2 m)$ update and query time, where m is a parameter between 1 and n.*

Proof: We consider the static case first. Partition H into $\lceil n/m \rceil$ groups $H_1, \ldots, H_{\lceil n/m \rceil}$, each of size at most m, compute the convex polygon $\Pi_i = \bigcap H_i$ for each i, and store each of them in an ordered array. The total preprocessing time is then $O(\frac{n}{m}(m \log m)) = O(n \log m)$, while space is linear. Reichling [38] showed that in $O(k \log^2 m)$ time, one can detect whether the intersection of k convex m-gons is empty, and if not, report the point in the intersection that is extreme in a given direction ξ; his method is based on Megiddo's prune-and-search technique. Using Reichling's algorithm on the $k = \lceil n/m \rceil$ polygons $\Pi_1, \ldots, \Pi_{\lceil n/m \rceil}$, we can answer a linear programming query in $O((n/m) \log^2 m)$ time.

The dynamic part can be proven using Overmars and van Leeuwen's data structure [34] to store each of the H_i's, which requires $O((n/m) \log^2 m)$ update time; Reichling's time bound still applies. □

As a result of this lemma, q linear programming queries in the plane can be answered in $O(n \log q)$ time for $q \leq n / \log n$.

Corollary 3.2 *A sequence of q linear programming queries and q insertions/deletions on a dynamic set H of at most n halfplanes in E^2 can be performed in $O(n \log q + q \log^2 n)$ time and $O(n)$ space.*

In higher dimensions, Matoušek [24] has obtained data structures for linear programming queries achieving the complexities shown in Table 1. His approach uses a multidimensional parametric search technique to reduce the problem of answering linear programming queries to that of answering halfspace-emptiness queries with witness; the halfspace-emptiness problem is then solved in the same way as ray shooting in polytopes. (In the dual setting, a *halfspace-emptiness query* on H is to determine whether a given query point p belongs to $\bigcap H$, and if not, provide a *witness* halfspace $h \in H$ that does not contain p; such a query can be performed by shooting the ray $\overrightarrow{op}$ in $\bigcap H$.)

We now show how to obtain a preprocessing time/query time tradeoff for the static Structure 1 in the case of linear programming. The bounds we get are similar to those obtained in Lemma 2.3, except for an extra polylogarithmic factor in n in the query time; this causes an additional $O(n \log \log n)$ term in the overall time bound.

Lemma 3.3 *There is a (static) data structure for linear programming queries on a set H of n halfspaces in E^d $(d > 2)$ with $O(n \log m)$ preprocessing time, $O(n)$ space, and $O((n/m^{1/\lfloor d/2 \rfloor}) \log^{O(1)} m \log^d n)$ query time, where m is a parameter between 1 and n.*

Proof: In this proof, we assume that the reader is familiar with Matoušek's technique [24].

We consider the halfspace-emptiness queries first. Partition H into $\lceil n/m \rceil$ groups $H_1, \ldots, H_{\lceil n/m \rceil}$, each of size at most m. For each of the H_i's, we build a data structure [24] with $O(m \log m)$ preprocessing time and $O(m)$ space, so that each halfspace-emptiness query on H_i can be answered in $O(\log m)$ parallel steps using $O(m^{1-1/\lfloor d/2 \rfloor} \log^{O(1)} m)$ processors. The total preprocessing time is then $O(\frac{n}{m}(m \log m)) = O(n \log m)$ and the space requirement remains linear. Since the halfspace-emptiness problem is decomposable, a query on H can be performed in $\tau(n, m) = O(\log m)$ parallel steps using $\pi(n, m) = O(\frac{n}{m}(m^{1-1/\lfloor d/2 \rfloor} \log^{O(1)} m)) = O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^{O(1)} m)$ processors; or sequentially, in $t(n, m) = O(\frac{n}{m}(m^{1-1/\lfloor d/2 \rfloor} \log^{O(1)} m)) = O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^{O(1)} m)$ time.

Matoušek has shown that any data structure for halfspace-emptiness queries (satisfying some reasonable conditions) can be used to answer linear programming queries by a multidimensional version of Megiddo's parametric search method [28]. The resulting query time is given by $O(t(n,m)\tau(n,m)^d \log^d \pi(n,m))$, which, in our case, is $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^{O(1)} m \log^d n)$. □

Corollary 3.4 *A sequence of q linear programming queries on a set H of n halfspaces in E^d $(d > 2)$ can be performed in $O(n \log\log n + n \log q + (nq)^{1-1/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n + q \log^{d+1} n)$ time.*

Proof: The proof is as in Corollary 2.4, except that for Case I $(q \leq n^{1/\lfloor d/2 \rfloor} / \log^K n)$ we use Lemma 3.3 with $m = (q \log^K n)^{\lfloor d/2 \rfloor}$ $(1 \leq m \leq n)$. The running time for Case I now becomes

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

□

Remark: Again, the complexity remains the same even if the value of q is not known in advance. (Use the sequence $q_i = (\log n)^{2^i}$ $(i = 1, 2, \ldots)$ for Case I.)

Since Matoušek's data structures can be made dynamic like the ray shooting structures (see the bottom half of Table 1), the following analogue of Lemma 2.7 is straightforward:

Lemma 3.5 *A sequence of q linear programming queries on a dynamic set H of at most n halfspaces in E^d $(d > 2)$ can be performed in*

i. $O(n \log\log n + n \log q + (nq)^{1-1/(\lfloor d/2 \rfloor+1)+\varepsilon} + qn^{1-2/(\lfloor d/2 \rfloor+1)+\varepsilon})$ time, if the number of insertions/deletions is $O(q)$;

ii. $O(n \log^2 n + (nq)^{1-1/(\lfloor d/2 \rfloor+1)+\varepsilon} + q \log^{d+1} n)$ time, if the number of insertions/deletions is $O(n)$.

Finally, we observe that for the semidynamic case, where there are no deletions, Lemma 3.5(ii) may be improved somewhat.

Lemma 3.6 *A sequence of q linear programming queries and n insertions on an initially empty set of halfspaces in E^d can be performed in $O(n \log^2 n + (nq)^{1-1/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n + q \log^{O(1)} n)$ time.*

Proof: As in the proof of Lemma 3.3, we consider the halfspace-emptiness problem first. Since, this problem is decomposable, the techniques by Bentley and Saxe [4] may be applied to convert a static structure to a semidynamic one (which increases building time and query time by a logarithmic factor). We then apply Matoušek's parametric search to use this structure for answering linear programming queries. The resulting time bound is only a polylogarithmic factor increase on the static bound in Corollary 3.4. □

Remark: The precise bound for the $O(q \log^{O(1)} n)$ term in Lemma 3.6 is $O(q \log^{d+2} n (\log\log n)^d)$, not $O(q \log^{d+2} n)$ as suggested in the extended abstract of this paper [5], since we have $\tau(n,m) = O(\log n)$, $\pi(n,m) = O(\log n)$, and $t(n,m) = O(\log^2 n)$, in Matoušek's notation, for the semidynamic version of the $O(n^{\lfloor d/2 \rfloor} \log^{O(1)} n)$-space halfspace-emptiness data structure. The $O(q \log^{d+2} n)$ bound is still correct, but in order to remove the unnecessary $\log\log n$ factors, we need to apply a multidimensional version of Cole's improved parametric search [14]. We will not go over this in detail, as for very large values of q the bound in Lemma 3.5(ii) is better anyway.

In the appendix, we show how to eliminate the $\log n$ factors in Lemma 3.3 and thus remove the $n \log\log n$ term from both Corollary 3.4 and Lemma 3.5(i), if we allow *randomization.*

4 Convex Hulls

We now show that the f-face convex hull of an n-point set can be constructed by performing $O(f)$ ray shooting queries in a polytope defined by n halfspaces. The algorithm we use is just the well-known gift-wrapping method [8, 36, 44] dualized, since a "gift-wrapping operation" corresponds to shooting a ray in the dual polytope. If the ray shooting queries are performed directly by scanning the halfspaces, then we get an $O(nf)$-time bound. We observe that this can be improved using the data structures from Section 2.

Theorem 4.1 *The convex hull of a set P of n points in E^d can be constructed in $O(n \log f + (nf)^{1-1/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n)$ time and $O(n + (nf)^{1-1/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n)$ space, where f is the number of hull faces.*

Proof: In the dual setting, our problem becomes computing an intersection of a set H of n halfspaces in E^d (assumed to be in general position), each containing the origin o. It suffices to compute the vertices of the intersection $\bigcap H$, from which one can easily generate the complete lattice structure of $\bigcap H$ in $O(f \log f)$ time by a dictionary.

First, an initial vertex v_0 can be found by performing d ray shooting queries in $\bigcap H$, since shooting a ray from o gives a point in a $(d-1)$-face, and shooting a ray from a point in a j-face inside its affine hull gives a point in a $(j-1)$-face $(1 \le j < d)$. Furthermore, given a vertex v, the vertices adjacent to v in the 1-skeleton (the graph formed by the vertices and edges of $\bigcap H$) can be found by performing d ray shooting queries: if $h_1, \ldots, h_d$ are the hyperplanes defining v, then shoot a ray from v along each of the d lines formed by intersecting $d-1$ hyperplanes from $\{h_1, \ldots, h_d\}$.

Since the 1-skeleton is connected, we can use a depth-first search (or a breadth-first search, or any graph traversal algorithm) to visit all vertices of $\bigcap H$; we can ensure that each vertex is visited only once by using a dictionary to detect replication. This shows that the vertices of $\bigcap H$ can be computed by performing $O(f)$ ray shooting queries in $\bigcap H$. The theorem then follows by applying Corollaries 2.2 and 2.4 (recall that $f = O(n^{\lfloor d/2 \rfloor})$). $\square$

5 Extreme Points

We now consider the problem of computing the h extreme points of an n-point set. Since determining whether a point is extreme can be done by solving a certain linear program, it is not difficult to see that n linear programming queries on n halfspaces are sufficient. We show that we can do better if h is small: by a simple algorithm, the extreme points can be found using h queries on n halfspaces together with n queries on h halfspaces. With Megiddo's linear programming algorithm [29], this leads to a simple $O(nh)$-time extrema algorithm. The same $O(nh)$-time algorithm has recently been discovered by Clarkson [12] and Ottmann et al. [32]. We note that the time bound can be further improved using the results from Section 3.

Theorem 5.1 *The h extreme points of a set P of n points in E^d can be computed in $O(n \log^{d+1} h + (nh)^{1-1/(\lfloor d/2 \rfloor+1)+\varepsilon})$ time or in $O(n \log^{O(1)} h + (nh)^{1-1/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n)$ time.*

Proof: Without loss of generality, assume that the origin o is in the interior of $\mathrm{conv}(P)$. Consider the following incremental algorithm, which is essentially the same as Clarkson's algorithm and the algorithm by Ottmann et al.:

Algorithm Extrema(P)

1. $Q \leftarrow \emptyset$
2. for each $p \in P$ (in any order) do
3.   if $p \notin Q$ and $p \notin \text{conv}(Q \cup \{o\})$ then
4.   if p is an extreme point of P then
5. $Q \leftarrow Q \cup \{p\}$
6.   else find the facet f of conv(P) that intersects ray $\overrightarrow{op}$
7.   let v be a vertex of f that is not in Q
8. $Q \leftarrow Q \cup \{v\}$
9. return Q

Observe that v must exist in line 7, because otherwise all vertices of f would be in Q; since $p \in \text{conv}(f \cup \{o\})$, this would imply that $p \in \text{conv}(Q \cup \{o\})$: a contradiction with line 3. It is then clear that the algorithm correctly returns the set of extreme points of P.

We now analyze the cost of the algorithm. Note that line 3 can be accomplished by solving a linear program on Q in the dual and lines 4 and 6 can be accomplished by solving a linear program on P in the dual. (Line 7 takes constant time since each facet has d vertices by the general position assumption.) Observe that although line 3 is executed n times, lines 4–8 are executed only h times since each execution adds a new point to Q. Thus, the algorithm requires h linear programming queries on P, a static set of size n, and n linear programming queries on Q, a semidynamic set of size at most h.

By Corollary 3.4, the h queries on P can be done in $O(n \log\log n + n \log h + (nh)^{1-1/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n)$ time. By Lemma 3.5(ii), the n queries and h insertions on Q can be done in $O((nh)^{1-1/(\lfloor d/2 \rfloor+1)+\varepsilon} + n \log^{d+1} h)$ time. The total running time is then $O(n \log\log n + n \log^{d+1} h + (nh)^{1-1/(\lfloor d/2 \rfloor+1)+\varepsilon})$.

Notice that when $h \leq n^{\alpha}$ for a constant $\alpha < (1/\lfloor d/2 \rfloor)^2$, the number of hull faces is $O(n^{1/\lfloor d/2 \rfloor - \varepsilon})$; so we can compute the entire convex hull in optimal $O(n \log h)$ time and $O(n)$ space by Theorem 4.1. This allows us to remove the $O(n \log\log n)$ term in the time bound. The first part of the theorem is thus proven, and the second part follows similarly, using Lemma 3.6 instead of Lemma 3.5(ii) for Q. $\square$

Theorem 5.1 has an interesting corollary. It implies a bound for the convex hull problem that is within a polylogarithmic factor of optimal in the worst case, if the complexity is measured in terms of n and the number of extreme points h. (Note that $\Omega(n \log h + h^{\lfloor d/2 \rfloor})$ is a lower bound in terms of n and h.)

Corollary 5.2 *The convex hull of a set P of n points in E^d can be constructed in $O(n \log^{O(1)} h + h^{\lfloor d/2 \rfloor})$ time, where h is the number of hull vertices.*

Proof: Compute the extreme points by Theorem 5.1 and then construct the convex hull of these h points by Chazelle's algorithm [10] in $O(h^{\lfloor d/2 \rfloor})$ time (note that when $h = \Omega(n^{1/\lfloor d/2 \rfloor})$, we have $h^{\lfloor d/2 \rfloor} = \Omega((nh)^{1-1/(\lfloor d/2 \rfloor+1)})$). $\square$

6 Convex Layers and Depths

We now consider the convex layers problem and the depth problem. For the depth problem, we use a hybrid of the methods of Sections 4 and 5 to obtain a subquadratic algorithm. Then we show how this leads to an output-sensitive convex layers algorithm using Seidel's convex hull algorithm [41].

Theorem 6.1 *The depth of all points in a set P of n points in E^d can be computed in $O(n^{2-\beta+\varepsilon})$ time, where $\beta = 2/(\lfloor d/2 \rfloor^2 + 1)$.*

Proof: We iteratively compute the vertices of the i-th layer ($i = 1, 2, \ldots$) as follows. We use the convex hull algorithm in Theorem 4.1 to construct the i-th layer, but as soon as more than n^β vertices are discovered in the layer, we stop the computation and switch to the extrema algorithm in Theorem 5.1 to compute the vertices of the layer. We then remove the vertices of the i-th layer from P and proceed to the $(i+1)$-st layer. In the end, we will have the depths of every point in P. For the calls to the convex hull algorithm, we will use a dynamic ray shooting data structure instead of a static one so that structures don't have to be rebuilt as points are removed from P after each iteration; for the calls to the extrema algorithm, however, we will leave the data structures unchanged.

Let h_i denote the number of vertices of the i-th layer ($\sum_i h_i = n$). We first analyze the cost of the calls to the convex hull algorithm in Theorem 4.1, which involve a number of ray shooting queries and n deletions on a dynamic set of at most n halfspaces; the number of queries is proportional to the number of facets discovered. Since we stop the computation in a layer when n^β vertices are found, we make at most $O(\min\{h_i^{\lfloor d/2 \rfloor}, n^{\beta \lfloor d/2 \rfloor}\})$ queries for the i-th layer. The total number of queries is then asymptotically bounded by

$$\begin{aligned}
\sum_{h_i \le n^\beta} h_i^{\lfloor d/2 \rfloor} + \sum_{h_i > n^\beta} n^{\beta \lfloor d/2 \rfloor} &\le n^{\beta(\lfloor d/2 \rfloor - 1)} \left(\sum_{h_i \le n^\beta} h_i \right) + n^{\beta(\lfloor d/2 \rfloor - 1)} \left(\sum_{h_i > n^\beta} n^\beta \right) \\
&\le n^{\beta(\lfloor d/2 \rfloor - 1)} \sum_i h_i \le n^{1+\beta(\lfloor d/2 \rfloor - 1)}.
\end{aligned}$$

By Lemma 2.6(ii), we see that the cost of these queries is $O((n^{2+\beta(\lfloor d/2 \rfloor - 1)})^{1-1/(\lfloor d/2 \rfloor + 1)+\varepsilon}) = O(n^{2-\beta+c\varepsilon})$ by our choice of β (where c is an appropriate constant).

Next, we analyze the cost of the calls to the extrema algorithm in Theorem 5.1. Note that the extrema algorithm is called only for the layers i with $h_i > n^\beta$, and the number of h_i's with $h_i > n^\beta$ is at most $n^{1-\beta}$ (since $\sum_i h_i = n$). Ignoring logarithmic factors, the cost is then

$$\begin{aligned}
\sum_{h_i > n^\beta} \left(n + (nh_i)^{1-1/(\lfloor d/2 \rfloor + 1)} \right) &\le n \left(\sum_{h_i > n^\beta} 1 \right) + n^{1-1/(\lfloor d/2 \rfloor + 1)} \left(\sum_{h_i > n^\beta} h_i^{1-1/(\lfloor d/2 \rfloor + 1)} \right) \\
&\le n\,(n^{1-\beta}) + n^{1-1/(\lfloor d/2 \rfloor + 1)} \left(\sum_{h_i > n^\beta} h_i \right)^{1-1/(\lfloor d/2 \rfloor + 1)} \left(\sum_{h_i > n^\beta} 1 \right)^{1/(\lfloor d/2 \rfloor + 1)} \\
&\le n^{2-\beta} + n^{1-1/(\lfloor d/2 \rfloor + 1)} \left(n^{1-1/(\lfloor d/2 \rfloor + 1)}\right) (n^{1-\beta})^{1/(\lfloor d/2 \rfloor + 1)} = O(n^{2-\beta}),
\end{aligned}$$

by Hölder's inequality. Therefore, the entire method runs in $O(n^{2-\beta+c\varepsilon})$ time. □

Corollary 6.2 *The convex layers of a set P of n points in E^d can be constructed in $O(n^{2-\beta+\varepsilon} + f \log n)$ time, where f is the total output size and $\beta = 2/(\lfloor d/2 \rfloor^2 + 1)$.*

Proof: Let P_i be the set of vertices of layer i (i.e., the points of depth i) and let h_i and f_i be the number of vertices and faces of the layer ($\sum_i h_i = n$, $\sum_i f_i = f$). We first compute P_i for all i in $O(n^{2-\beta+\varepsilon})$ time by Theorem 6.1 and then construct the convex hull of each P_i using Seidel's algorithm [41] with Matoušek's improvement [24]. The total time needed is $O(n^{2-\beta+\varepsilon} + \sum_i (h_i^{2-2/(\lfloor d/2 \rfloor+1)+\varepsilon} + f_i \log h_i)) = O(n^{2-\beta+\varepsilon} + f \log n)$. □

Remarks:

1. A worst-case optimal convex layers algorithm for $d \geq 4$ is not difficult to get: just use an $O(nh)$-time extrema algorithm to compute the vertices of each layer and use Chazelle's convex hull algorithm [10] to construct the layers; then the running time is $O(n^2 + n^{\lfloor d/2 \rfloor})$.

2. For a more direct output-sensitive convex layers algorithm, we can simply do the following: iteratively use the convex hull algorithm in Theorem 4.1 to construct the i-th layer ($i = 1, 2, \ldots$) and delete points from P that are vertices of a layer after each iteration. This method is the same as the one by Agarwal and Matoušek [2] for the three-dimensional case. It requires $O(f)$ ray shooting queries and n deletions, and by Lemma 2.7(ii), takes $O((nf)^{1-1/(\lfloor d/2 \rfloor+1)+\varepsilon})$ time, which is superior to the bound in Corollary 6.2 only when f is near linear (recall $\Omega(n) = f = O(n^{\lfloor d/2 \rfloor})$).

7 Other Applications

We now consider applications of our techniques to the construction of a k-level in an arrangement and to linear programming with few violated constraints.

Theorem 7.1 *A k-level in an arrangement $\mathcal{A}(H)$ of n hyperplanes in E^d can be constructed in*

i. $O(n \log f + f \log^2 n)$ *time, if* $d = 2$*;*

ii. $O(n \log f + f^{1+\varepsilon})$ *time, if* $d = 3$*;*

iii. $O(n \log f + (nf)^{1-1/(\lfloor d/2 \rfloor+1)+\varepsilon} + f n^{1-2/(\lfloor d/2 \rfloor+1)+\varepsilon})$ *time, if* $d \geq 4$*;*

where f is the output size.

Proof: The depth-first search algorithm by Agarwal and Matoušek [2] constructs the k-level using $O(f)$ polytope ray shooting queries and $O(f)$ insertions/deletions on two dynamic sets of at most n halfspaces. (In two dimensions, their algorithm is the same as Edelsbrunner and Welzl's [19].) Hence, the theorem follows from Corollary 2.6 and Lemma 2.7(i). □

Theorem 7.2 *The linear programming problem on n constraints in E^d with at most k violations for the feasible case can be solved in*

i. $O(n \log k + k^2 \log^2 n)$ *time, if* $d = 2$*;*

ii. $O(n \log \log n + n \log k + k^{3+\varepsilon})$ *time, if* $d = 3$*;*

iii. $O(n \log\log n + n \log k)$ *time, if* $d \geq 4$ *and* $k^d \leq n^{1/\lfloor d/2 \rfloor - \varepsilon}$.

Proof: The depth-first search algorithm by Matoušek [25] solves this problem using $O(k^d)$ linear programming/membership queries and $O(k^d)$ insertions/deletions on two dynamic sets of at most n halfspaces. (A membership query is just a special case of a ray shooting query.) Hence, part (i) of the theorem follows from Corollaries 2.6 and 3.2, and parts (ii) and (iii) follow from Lemmas 2.7(i) and 3.5(i). With randomization, the results from the appendix can even eliminate the $n \log\log n$ terms in parts (ii) and (iii). □

Remark: For large values of k in the two-dimensional case, the time bound of Theorem 7.2(i) can be reduced to $O(n \log^2 n)$ using parametric search or slope selection techniques, as Matoušek [25] and Roos and Widmayer [40] observed.

The techniques here may also be applicable to the infeasible case of linear programming with k violated constraints, or to the smallest k-enclosing circle problem; see Matoušek's paper [25].

Finally, we mention an improvement to Mulmuley's output-sensitive algorithm [30] for constructing $(\leq k)$-levels. The algorithm assumes that the input hyperplanes H are *nonredundant*, i.e., every hyperplane in H supports the upper envelope of H. For applications to $(\leq k)$-order Voronoi diagrams, this assumption is automatically satisfied.

Theorem 7.3 *We can compute i-levels in an arrangement* $\mathcal{A}(H)$ *of* n *nonredundant hyperplanes in* E^d *for all* $i = 0, 1, \ldots, k$ *in* $O(n^{2-2/(\lfloor d/2 \rfloor + 1)+\varepsilon} k^{d-1} + f \log n)$ *time, where* f *is the output size.*

Proof: Let $L_i(H)$ denote the boundary of the i-level in $\mathcal{A}(H)$ and let f_i be its size ($\sum_{i=0}^{k} f_i = f$). For each $h \in H$, let $H_h = \{h \cap h' : h' \in H - \{h\}\}$, which is a set of $(d-1)$-dimensional hyperplanes.

Mulmuley [30] gave an algorithm which constructs the facial structure of $L_i(H)$ in $O((f_i + f_{i-1}) \log n)$ time, given the following information:

1. the local minima (along some predefined direction) of the i-level in $\mathcal{A}(H)$ that lie on $L_i(H) - L_{i-1}(H)$;
2. the local minima of the i-level in $\mathcal{A}(H_h)$ that lie on $L_i(H_h) - L_{i-1}(H_h)$, for each $h \in H$; and
3. the facial structure of $L_{i-1}(H)$.

Matoušek [25] has shown that the local minima of all i-levels in $\mathcal{A}(H)$ ($i = 0, 1, \ldots, k$) can be enumerated by performing $O(k^d)$ linear programming/membership queries and $O(k^d)$ insertions/deletions on two dynamic sets of at most n halfspaces. Similarly, the local minima of all i-levels in $\mathcal{A}(H_h)$ ($i = 0, 1, \ldots, k$) can be computed using $O(k^{d-1})$ linear programming/membership queries and $O(k^{d-1})$ insertions/deletions, for each $h \in H$. Observe that we do not need separate structures to store each H_h as the data structures from Section 3 can perform linear programming queries restricted to any j-flat [24]. The total number of queries and updates is then $O(k^d + nk^{d-1}) = O(nk^{d-1})$. By Lemmas 2.7(i) and 3.5(i), this takes $O(n^{2-2/(\lfloor d/2 \rfloor + 1)+\varepsilon} k^{d-1})$ time.

Thus, items 1 and 2, for all $i = 0, 1, \ldots, k$, can be computed in $O(n^{2-2/(\lfloor d/2 \rfloor + 1)+\varepsilon} k^{d-1})$ time. Now, Mulmuley's algorithm can be used to construct the facial structure of $L_0(H), L_1(H), \ldots, L_k(H)$ incrementally, in additional $O(f \log n)$ time. □

8 Final Remarks

We remark that further applications of our ideas are possible. For example, Theorem 4.1 can be extended to compute the intersection of a convex hull with a j-flat in an output-sensitive manner; in the dual, this corresponds to computing projections (shadows) of an intersection of halfspaces. More generally, we can obtain output-sensitive bounds for computing "skeletons" in a halfspace intersection, or with the known methods for ray shooting in a collection of hyperplanes [1], "skeletons" in a hyperplane arrangement; see Edelsbrunner [16, Chapter 9]. With suitable data structures, this applies to arrangements of different objects as well, such as line segments in the plane.

Many open questions remain, however. A major problem is to find an $O((n+f)^{1+\varepsilon})$-time convex hull algorithm in dimensions higher than four. Another question is: can the depth problem be solved in $O(n^{2-2/(\lfloor d/2\rfloor+1)+\varepsilon})$ time?

Acknowledgements

I am grateful to Jack Snoeyink for his guidance and encouragement as well as for many valuable suggestions and stimulating discussions. I would also like to thank the referees for pointing out Clarkson's result on extreme points.

Appendix. Using Randomization in Linear Programming Queries

Consider the following linear programming problem: given k preprocessed polytopes $\Pi_1, \ldots, \Pi_k \subset E^d$, each defined by m halfspaces containing the origin o, compute the vertex v of $\Pi_1 \cap \ldots \cap \Pi_k$ that maximizes $\xi \cdot v$ for a given $\xi \in E^d$. Suppose that linear-space static structures (Structure 1) from Table 1 are used to store these polytopes. As is demonstrated in the proof of Lemma 3.3, a direct application of Matoušek's multidimensional parametric search technique would yield an $O(k\, m^{1-1/\lfloor d/2\rfloor} \log^{O(1)} m \log^d k)$-time solution. Here we describe how the $\log^d k$ factor can be eliminated by using Sharir and Welzl's randomized algorithm for generalized linear programming [43] (which is based on Seidel's linear programming algorithm [42]). This in turn improves the query time in Lemma 3.3.

We first observe that the problem of finding an extremum in a nonempty intersection of k convex objects in E^d belongs to the class of *LP-type problems of combinatorial dimension* d as defined by Sharir and Welzl [43]. Sharir and Welzl presented a simple randomized algorithm for solving LP-type problems of fixed combinatorial dimension that requires an expected number of $O(k)$ *primitive operations*. The primitive operations, in our case, are: (i) to test whether a given point lies inside one of the objects (*violation tests*), and (ii) to find the extremum in an intersection of $d+1$ of the objects (*basis computations*).

For our application, the objects are (convex) polytopes. A violation test is simply a membership query and costs $O(m^{1-1/\lfloor d/2\rfloor} \log^{O(1)} m)$ time. A basis computation involves solving our linear programming problem on $d+1$ of the polytopes; since the number of polytopes is now constant, we can apply our previous method, via Matoušek's parametric search, to solve this problem in $O(m^{1-1/\lfloor d/2\rfloor} \log^{O(1)} m)$ time. Because $O(k)$ violation tests and basis computations are expected to be performed by Sharir and Welzl's algorithm (the expected number of basis computations is actually only $O(\log^d k)$ [45]), we obtain a randomized $O(k\, m^{1-1/\lfloor d/2\rfloor} \log^{O(1)} m)$-time solution to our linear programming problem on k polytopes.

The above method carries through if the polytopes are stored in linear-space dynamic structures (Structure 1$'$ from Table 1); we simply replace the $\log^{O(1)} m$ factors with m^ε. With slightly more effort, we can even remove the assumption that the polytopes all contain the origin; the method can detect whether k preprocessed polytopes have a common intersection.

Note that in the two-dimensional case both violation tests and basis computations can be performed in $O(\log m)$ time. Thus, Sharir and Welzl's algorithm achieves expected $O(k \log m)$ time, which is an improvement over the previous $O(k \log^2 m)$ algorithm by Reichling [38], as used in our proof of Lemma 3.1. It is also interesting to compare the techniques here with those used in the previous deterministic and randomized methods by Reichling [39] and Eppstein [21] for the three-dimensional problem.

The (expected) query time in Lemma 3.3 can now be improved to $O((n/m^{\lfloor d/2\rfloor}) \log^{O(1)} m)$ since it uses $k = \lceil n/m \rceil$. As a consequence, the $O(n \log\log n)$ term in Corollary 3.4 can be eliminated; the same is true for the dynamic case (Lemma 3.5(i)), which leads to corresponding improvements in Theorem 7.2.

References

[1] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22:794–806, 1993.

[2] P. K. Agarwal and J. Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13:325–345, 1995.

[3] J. L. Bentley, H. T. Kung, M. Schkolnick and C. D. Thompson. On the average number of maxima in a set of vectors. *J. ACM*, 25:536–543, 1978.

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

[5] T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. In *Proc. 11th ACM Sympos. Comput. Geom.*, 10–19, 1995.

[6] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete Comput. Geom.*, this issue.

[7] T. M. Chan, J. Snoeyink, and C.-K. Yap. Output-sensitive construction of polytopes in four dimensions and clipped Voronoi diagrams in three. In *Proc. 6th ACM-SIAM Sympos. on Discrete Algorithms*, 282–291, 1995.

[8] D. R. Chand and S. S. Kapur. An algorithm for convex polytopes. *J. ACM*, 17:78–86, 1970.

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

[10] B. Chazelle. An optimal convex hull algorithm for point sets in any fixed dimension. *Discrete Comput. Geom.*, 9:145–158, 1993.

[11] B. Chazelle and J. Matoušek. Derandomizing an output-sensitive convex hull algorithm in three dimensions. *Comput. Geom. Theory Appl.*, 5:27–32, 1995.

[12] K. L. Clarkson. More output-sensitive geometric algorithms. In *Proc. 35th IEEE Sympos. Found. of Comput. Sci.*, 695–702, 1994.

[13] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[14] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34:200–208, 1987.

[15] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Comput. Sci.*, 27:241–253, 1983.

[16] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin, 1987.

[17] H. Edelsbrunner and W. Shi. An $O(n \log^2 h)$ time algorithm for the three-dimensional convex hull problem. *SIAM J. Comput.*, 20:259–277, 1991.

[18] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graphics*, 9:66–104, 1990.

[19] H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrangements with applications. *SIAM J. Comput.*, 15:271–284, 1986.

[20] I. Emiris and J. Canny. An efficient approach to removing geometric degeneracies. In *Proc. 8th ACM Sympos. Comput. Geom.*, 74–82, 1992.

[21] D. Eppstein. Dynamic three-dimensional linear programming. In *Proc. 32nd IEEE Sympos. Found. of Comput. Sci.*, 488–494, 1991.

[22] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1:132–133, 1972.

[23] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15:287–299, 1986.

[24] J. Matoušek. Linear optimization queries. *J. Algorithms*, 14:432–448, 1993. Also with O. Schwarzkopf in *Proc. 8th ACM Sympos. Comput. Geom.*, 16–25, 1992.

[25] J. Matoušek. On geometric optimization with few violated constraints. In *Proc. 10th ACM Sympos. Comput. Geom.*, 312–321, 1994.

[26] J. Matoušek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete Comput. Geom.*, 10:215–232, 1993.

[27] P. McMullen. The maximal number of faces of a convex polytope. *Mathematika*, 17:179–184, 1970.

[28] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30:852–865, 1983.

[29] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31:114–127, 1984.

[30] K. Mulmuley. Output sensitive construction of levels and Voronoi diagrams in R^d of order 1 to k. In *Proc. 22nd ACM Sympos. Theory of Comput.*, 322–330, 1990.

[31] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, New York, 1993.

[32] T. Ottmann, S. Schuierer, and S. Soundaralakshmi. Enumerating extreme points in higher dimensions. In *Proc. 12th Sympos. Theoretical Aspects of Comput. Sci.*, Lect. Notes in Comput. Sci., vol. 900, 562–570, 1995.

[33] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.

[34] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[35] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20:87–93, 1977.

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

[37] H. Raynaud. Sur l'enveloppe convexe des nuages des points aléatoires dans R^n. *J. Appl. Probab.*, 7:35–48, 1970.

[38] M. Reichling. On the detection of a common intersection of k convex objects in the plane. *Inform. Process. Lett.*, 29:25–29, 1988.

[39] M. Reichling. On the detection of a common intersection of k convex polyhedra. In *Computational Geometry and its Applications*, Lect. Notes in Comput. Sci., vol. 333, 180–186, 1988.

[40] T. Roos and P. Widmayer. k-Violation linear programming. *Inform. Process. Lett.*, 52:109–114, 1994.

[41] R. Seidel. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proc. 18th ACM Sympos. Theory of Comput.*, 404–413, 1986.

[42] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete Comput. Geom.*, 6:423–434, 1991.

[43] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proc. 9th Sympos. Theoretical Aspects of Comput. Sci.*, Lect. Notes in Comput. Sci., vol. 577, 569–579, 1992.

[44] G. F. Swart. Finding the convex hull facet by facet. *J. Algorithms*, 6:17–48, 1985.

[45] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer (Ed.), *New Results and New Trends in Computer Science*, Lect. Notes in Comput. Sci., vol. 555, 359–370, 1991.