CS-1993-20

An Efficient Multi-Dimensional Searching Technique and its Applications

Pankaj K. Agarwal   Micha Sharir
Sivan Toledo

Department of Computer Science

Duke University

Durham, North Carolina 27708–0129

August 16, 1993

An Efficient Multi-Dimensional Searching Technique and its Applications *

Pankaj K. Agarwal[†] Micha Sharir[‡] Sivan Toledo[§]

August 16, 1993

Abstract

This paper describes an improved algorithm for the multi-dimensional searching problem introduced by Megiddo. As a result, we obtain a $d^{O(d)}n$ time deterministic algorithms for linear programming in $\mathbb{R}^d$ with n constraints, for computing the Euclidean 1-center of a set of n points in $\mathbb{R}^d$, for computing the minimum enclosing ellipsoid of a set of n points in $\mathbb{R}^d$, etc. Our techniques also improve the running time of known algorithms for a number of parametric graph searching problems, including that of finding zero cycles in dynamic graphs [7].

1 Introduction

Let h be an oriented hyperplane in $\mathbb{R}^d$ defined as

$$h : a_1x_1 + \cdots + a_{d-1}x_{d-1} + a_dx_d = b\,,$$

where $(a_1, \ldots, a_d)$ is the direction of the positive normal to h. With a slight abuse of notation we will also use h to denote a linear function $h : \mathbb{R}^d \to \mathbb{R}$ defined by $h(\mathbf{x}) = a_1x_1 + \cdots + a_{d-1}x_{d-1} + a_dx_d - b$. For a nonempty convex set $\Lambda \subseteq \mathbb{R}^d$, we define a *sign*

*Pankaj Agarwal has been supported by National Science Foundation Grant CCR-91-06514. Micha Sharir has been supported by Office of Naval Research Grant N00014-90-J-1284, by National Science Foundation Grant CCR-89-01484, and by grants from the U.S.–Israeli Binational Science Foundation, the G.I.F. — the German Israeli Foundation for Scientific Research and Development, and the Fund for Basic Research administered by the Israeli Academy of Sciences. Sivan Toledo has been supported by the Defense Advanced Research Projects Agency under Grant N00014–91–J–1698.

†Department of Computer Science, Duke University.

‡Courant Institute of Mathematical Sciences, New York University, and School of Mathematical Sciences, Tel Aviv University.

§Laboratory for Computer Science, Massachusetts Institute of Technology.

function $\text{sign}_\Lambda(h)$ as

$$\text{sign}_\Lambda(h) = \begin{cases} 0 & h(\mathbf{x}) = 0 \text{ for some } \mathbf{x} \in \Lambda, \\ +1 & h(\mathbf{x}) > 0 \text{ for all } \mathbf{x} \in \Lambda, \\ -1 & h(\mathbf{x}) < 0 \text{ for all } \mathbf{x} \in \Lambda. \end{cases}$$

Let $\mathcal{O} = \mathcal{O}(\Lambda)$ be an oracle that, given an oriented hyperplane h, computes $\text{sign}_\Lambda(h)$. In this paper we consider the following problem:

> *Given a set $H = \{h_1, \ldots, h_n\}$ of n oriented hyperplanes in $\mathbb{R}^d$ and a convex set Λ with an associated oracle $\mathcal{O}$ as defined above, either return a hyperplane, not necessarily a hyperplane of H, that intersects Λ, or compute* $\text{sign}_\Lambda(h_i)$ *for every $1 \leq i \leq n$.*

The goal is to minimize the number of oracle calls. This problem arises in the multi-dimensional searching technique, originally introduced by Megiddo [23] to obtain a linear time algorithm for linear programming in $\mathbb{R}^d$, where d is a constant. This technique has also found applications in a wide variety of other convex optimization problems, including the Euclidean 1-center problem [11], computing minimum enclosing ellipsoids [12], detecting minimum zero cycles in dynamic graphs [7], and general transshipment problems [25].

Megiddo showed that $\text{sign}(h)$, for all $h \in H$, can be computed by calling the oracle $2^{O(2^d)} \log n$ times,[1] which was later independently improved by Clarkson [5] and Dyer [11] to $O(2^d \log n)$. Although in the original paper Megiddo assumed Λ to be a point, his technique can easily be extended to an arbitrary convex set; see e.g. [7, 9]. Using the algorithm for resolving the signs of hyperplanes, Megiddo presented an algorithm for linear programming in $\mathbb{R}^d$ with n constraints, whose running time is $n \cdot 2^{O(2^d)}$, which can be reduced to $O(n \cdot 3^{d^2})$ by using the improved technique of Clarkson and Dyer. Dyer [11] showed that the same technique can be applied to solve the Euclidean 1-center problem (i.e., finding the smallest enclosing ball containing a set of n points) with the same asymptotic running time; see also [12] for some recent results.

In this paper we present a deterministic algorithm that either returns a hyperplane that intersects Λ, or resolves the signs of all hyperplanes in H, by making $O(d^3 \log d \log n)$ oracle calls, which is a significant improvement over the previous algorithms. Using this algorithm as a subroutine, we present a deterministic algorithm for linear programming in $\mathbb{R}^d$ with n constraints, whose worst case running time is $n \cdot O(d)^{10d} \log^{2d} d$. A similar approach yields algorithms with improved dependence on d for other geometric optimization problems mentioned above. Recently Chazelle and Matoušek [4] have also developed a deterministic algorithm for linear programming, using a different approach, whose worst case running time is $n \cdot O(d)^{7d(1+o(1))}$. Their approach however does not extend to computing the general

[1] Throughout this paper the constants of proportionality hidden in the big-Oh notation are *independent* of d, and the base of logarithms is 2.

multi-dimensional searching Like all the previous approaches, our procedure for computing the signs of hyperplanes works in $O(\log n)$ stages, each stage resolving the signs of at least a fraction of hyperplanes. More specifically, we present an algorithm that, by making at most $O(d^3 \log d)$ oracle calls, either finds a hyperplane intersecting Λ, or resolves the signs of at least half of the hyperplanes in H.

We also consider a somewhat more general problem. Let $w : H \to \mathbb{R}^+$ be an arbitrary *weight function*. For a subset $N \subseteq H$, define $w(N) = \sum_{h \in N} w(h)$.

> *Given a convex set Λ with an associated oracle $\mathcal{O}$ as defined above, a set H of oriented hyperplanes, and a weight function $w : H \to \mathbb{R}^+$, either return a hyperplane, not necessarily a hyperplane of H, that intersects Λ, or compute the signs of a subset $N \subseteq H$ such that $w(N) \geq w(H)/2$.*

This weighted version of the problem arises in some variants of multi-dimensional searching. For example, Cole [10] showed that, by assigning weights to lines whose signs have to be resolved, the running time of Megiddo's (one dimensional) parametric search technique in certain special cases can be improved by a logarithmic factor. Recently, Agarwala and Fernández-Baca [1] extended Megiddo's original algorithm for computing signs of hyperplanes to the weighted case, and showed that, using Cole's technique, one can improve the running time of the multi-dimensional searching by a factor of $\log^d n$ in some special cases. However, their algorithm is rather involved. We will show that our algorithm for the unweighted case can be extended to a set of weighted hyperplanes without any significant additional work, and thus gives a faster algorithm for the weighted version of the problem.

The paper is organized as follows. In Section 2 we present various geometric concepts needed by our algorithm. Section 3 describes the algorithm for resolving the signs of a set of hyperplanes, and Section 4 discusses various applications, including linear programming and some parametric graph searching problems.

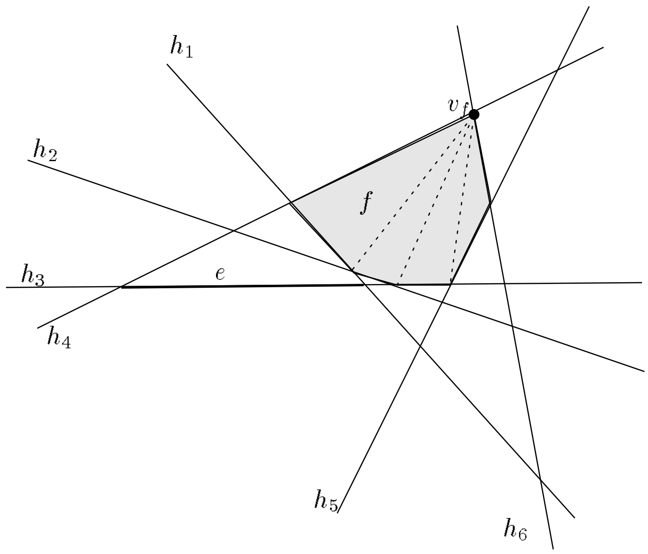
2 Geometric Preliminaries

In this section we describe certain concepts and results related to arrangements of hyperplanes and to ϵ-nets, which will be used later in the paper.

2.1 Arrangements of hyperplanes

Let H be a set of n hyperplanes in $\mathbb{R}^d$. H defines a decomposition of $\mathbb{R}^d$ into maximal connected relatively open regions of various dimensions, called *faces*, such that all points within each face lie on the same set of hyperplanes of H (and avoid all other hyperplanes). This decomposition is called the *arrangement* of H, and is denoted by $\mathcal{A}(H)$.

If the hyperplanes in H are in *general position*, then each k-dimensional face (or k-face for brevity) lies in exactly $d-k$ hyperplanes of H; see [13] for details on arrangements. Each face f of $\mathcal{A}(H)$ is a convex set, and can be implicitly represented by the sequence $\sigma(f) = (\mathrm{sign}_f(h_1), \mathrm{sign}_f(h_2), \ldots, \mathrm{sign}_f(h_n))$, called the *sign sequence* of f. The sign sequence of each face is unique. If f is a d-dimensional face, then $\sigma(f)$ consists of only $+1$'s and -1's.



$$\begin{aligned}\sigma(f) &= (+1,+1,+1,-1,+1,-1)\\ \sigma(e) &= (-1,-1,0,-1,+1,-1)\end{aligned}$$

Figure 1: An arrangement of lines, sign sequences of a face and an edge, and the canonical triangulation of a 2-face f; for the sign sequences, the positive sides of all lines are assumed to be their upper sides

The *canonical triangulation* of a k-face f of $\mathcal{A}(H)$, denoted as $CT(f)$, is a decomposition of $\mathcal{A}(H)$ into a collection of (closed) k-simplices with disjoint interiors, defined inductively as follows: If $k = 0, 1$, then $CT(f)$ consists of only one simplex, namely f itself. Suppose we have already defined the canonical triangulation of all j-faces for $j < k$. Let f be a k-face, and let v_f be the maximum vertex of f in reverse lexicographical order;[2] we will refer to v_f

[2]The *reverse lexicographical* ordering $(\underset{\mathrm{rlex}}{>})$ is defined as follows: Let $p = (p_1, p_2, \ldots, p_d)$, $q = (q_1, q_2, , \ldots, q_d)$ be two points in $\mathbb{R}^d$. Then $p \underset{\mathrm{rlex}}{>} q$ if there exists $i \le d$ such that $p_j = q_j$ for all $j > i$ and $p_i < q_i$.

as the *topmost* vertex of f. $CT(f)$ consists of simplices formed by connecting v_f to every $(k-1)$-simplex of $CT(g)$, for each $(k-1)$-face g incident upon f but not adjacent to v_f. That is,

$$CT(f) = \{CH(\triangle \cup \{v_f\}) \mid \triangle \in CT(g) \text{ s.t. } v_f \notin g \text{ and } g \text{ is a } (k-1)\text{-face of } f\}.$$

If f is unbounded we add a $(d-1)$-dimensional hyperplane at infinity, and compute the canonical triangulation of the new virtually trimmed k-face. For each simplex $\triangle \in CT(f)$, we define $V(\triangle)$ to be the sequence $v_1 \underset{\text{rlex}}{>} v_2 \underset{\text{rlex}}{>} \cdots \underset{\text{rlex}}{>} v_{k+1}$ of vertices of $\triangle$. Notice that $CH(\{v_{k-i+1}, \ldots, v_{k+1}\})$ is supported by a i-face of $\mathcal{A}(H)$ incident upon f.

2.2 Range spaces and ϵ-nets

Next, we review some definitions and properties of *range spaces* and *ϵ-nets*, taken from Haussler and Welzl [15] and from Matoušek [20]. A *range space* $\Sigma = (X, \mathcal{R})$ is a pair, where X is set and $\mathcal{R} \subseteq 2^X$. X is generally called a set of *objects* and $\mathcal{R}$ a set of *ranges*. The *subrange space* induced by a subset $Y \subseteq X$ is $(Y, \{Y \cap R \mid R \in \mathcal{R}\})$. A subset $Y \subseteq X$ is said to be *shattered* by Σ if $\{Y \cap R \mid R \in \mathcal{R}\} = 2^Y$. The *Vapnik-Chervonenkis dimension* (or *VC-dimension* for brevity) of Σ is the maximum size of a shattered subset of X (it is $+\infty$ if no such maximal size exists). The *shatter function*, denoted as $\pi_\Sigma(m)$, of Σ is the maximum possible number of ranges in a subrange space of Σ induced by a subset of X of size m. If the VC-dimension of Σ is $\delta < +\infty$, then $\pi_\Sigma(m) = O(m^\delta)$ [15].

If Σ is a finite range space (i.e., X is finite), then a subset $N \subseteq X$ is called an *ϵ-net*, for a parameter $0 < \epsilon < 1$, if $N \cap R \neq \emptyset$ for every $R \in \mathcal{R}$ with $|R| > \epsilon|X|$. Haussler and Welzl [15] showed that range spaces with VC-dimension δ have ϵ-nets of size $O(\frac{\delta}{\epsilon}\log\frac{\delta}{\epsilon})$. Notice that the size of an ϵ-net for range spaces with finite VC-dimension depends only on δ and ϵ, and not on $|X|$. Let $w : X \to \mathbb{R}^+$ be a weight function and, for a subset $A \subseteq X$, let $w(A) = \sum_{x \in X} w(x)$. Then a subset $N \subseteq X$ is called an ϵ-net for the weighted range space (Σ, w) if and only if $N \cap R \neq \emptyset$ for every $R \in \mathcal{R}$ with $w(R) > \epsilon w(X)$.

We will be using the following result of Matoušek [20], which has been simplified by Chazelle and Matoušek [4].

Theorem 2.1 (Chazelle-Matoušek [4]) *Let $\Sigma = (X, \mathcal{R})$ be a finite range space and let $r > 1$ be a parameter. Suppose that $\pi_\Sigma(m) = O(m^\delta)$ and that there is an algorithm, which, given a subset $N \subseteq X$, can compute in time $O(|N|^{\delta+1})$ the ranges of the subspace of Σ induced by N. Then one can compute, in time $O(\delta)^{3\delta} r^{2\delta} \log^\delta(\delta r) \cdot |X|$, a $(1/r)$-net of size $O(\delta r \log \delta r)$ for Σ.*

Consider the following range space H_d for a set H of n hyperplanes in $\mathbb{R}^d$:

$$H_d = (H, \{\, \{h \in H \mid h \cap \gamma \neq \emptyset\} \mid \gamma \text{ is a segment in } \mathbb{R}^d \}). \tag{2.1}$$

Let $G \subseteq H$ be a set of m hyperplanes in $\mathbb{R}^d$. Then, for every pair of faces f_1, f_2 in $\mathcal{A}(G)$, the segments $\overline{pq}$ such that $p \in f_1, q \in f_2$ intersect the same set of hyperplanes of G, so it is easily seen that $\pi_{H_d}(m) = O(m^{2d})$ and that the ranges of H_d induced by G can be computed in time $O(m^{2d+1})$. Hence, by Theorem 2.1, we obtain

Corollary 2.2 *Let H_d be the range space as defined in (2.1), and let $1 < r < n$ be a parameter. Then one can compute, in time $n \cdot O(d)^{6d} r^{4d} \log^{2d}(dr)$, a $(1/r)$-net of size $O(dr \log dr)$ for H_d.*

Following the approach of Matoušek [19], one can compute a $(1/r)$-net for a weighted range space (H_d, w) as well. The basic idea is as follows. First normalize the weight function so that $w(H) = n$, and then construct a multiset H' in which a hyperplane $h \in H$ has $\lceil w(h) \rceil$ copies. It is easily seen that H' has at most $2n$ hyperplanes. Next, compute a $(1/(2r))$-net N for the range space H'_d. It can be checked that the set of hyperplanes in N form a $(1/r)$-net for (H_d, w). See [19] for details.

Corollary 2.3 *Let H be a set of n hyperplanes in $\mathbb{R}^d$, let $w : H \to \mathbb{R}^+$ be a given weight function, and let $1 < r < n$ be a parameter. One can compute, in time $n \cdot O(d)^{6d} r^{4d} \log^{2d}(dr)$, a $(1/r)$-net of size $O(dr \log dr)$ for the weighted range space (H_d, w).*

3 Resolving the Signs of Hyperplanes

In this section we present an algorithm for resolving the signs of a set of hyperplanes with respect to a convex set Λ in $\mathbb{R}^d$. That is, given an oracle $\mathcal{O}$ for Λ, and a set H of n hyperplanes, either return a hyperplane (not necessarily a hyperplane of H) intersecting Λ, or compute $\mathrm{sign}_\Lambda(h)$ for all $h \in H$. As mentioned in the Introduction, we actually present a deterministic algorithm that either returns a hyperplane intersecting Λ, or resolves the signs of at least half of the hyperplanes in H.

Let r be some constant to be chosen later. We compute a $(1/r)$-net $G = \{h_1, \ldots, h_t\}$ of size $t = O(dr \log dr)$ for the range space H_d defined in (2.1). For the sake of simplicity, we assume that the hyperplanes in G are in general position, so the intersection of any k hyperplanes in G is a $(d-k)$-flat. Later we will describe how to modify the algorithm so that it works even if the hyperplanes in G are not in general position. Our technique first computes a simplex $\triangle \in CT(C)$, for some cell C of $\mathcal{A}(G)$, that fully contains Λ; if no such simplex exists, the algorithm will be able to return a hyperplane intersecting Λ.

Here is the basic idea behind computing $\triangle$: We first compute the cell C of $\mathcal{A}(G)$ that contains Λ by calling the oracle for each $h \in G$. Let v be the topmost vertex of C, and let ρ be some ray emanating from v that intersects Λ. We compute the first hyperplane $h \in G$ intersected by ρ. We, of course, do not know the ray ρ, so use Megiddo's parametric search technique to compute h. This step calls the oracle at most $t-1$ times. We recursively find the simplex $\triangle_h$ in the canonical triangulation of the arrangement of $\{h \cap h' \mid h' \in G - \{h\}\}$, that contains the projection of Λ onto h with respect to v, and set $\triangle = CH(\triangle_h \cup \{v\})$.

We now describe the procedure in detail. The algorithm computes $V(\triangle) = (v_1, v_2, \ldots, v_{d+1})$ in $d+1$ steps. If it has not returned a hyperplane intersecting Λ in the first i steps, then after the i^{th} step we have a set $V^{(i)} = (v_1, v_2, \ldots, v_i)$ of i vertices, such that $v_j \underset{\text{rlex}}{>} v_{j+1}$ for $1 \le j < i$, and a subset $\Psi^{(i)} = \{\psi_1, \ldots, \psi_i\} \subseteq G$ of hyperplanes. Let $\gamma^{(i)}$ be the $(d-i)$-flat formed by the intersection of the hyperplanes of $\Psi^{(i)}$. For the sake of convenience we regard $\gamma^{(i)}$ as $\mathbb{R}^{d-i}$. The definition of the sign function can be extended to hyperplanes lying in $\gamma^{(i)}$ in a natural way. Initially $V^{(0)} = \Psi^{(0)} = \emptyset$ and $\gamma^{(0)} = \mathbb{R}^d$. We define $\Lambda^{(i)}$, the *projection* of Λ on $\gamma^{(i)}$, as follows: $\Lambda^{(0)} = \Lambda$ and $\Lambda^{(i)}$ is the projection of $\Lambda^{(i-1)}$ on $\gamma^{(i)}$ with respect to v_i, i.e., if we denote by $Cone^{(i)}$ the set of rays emanating from v_i and intersecting $\Lambda^{(i-1)}$, then

$$\Lambda^{(i)} = \{\rho \cap \gamma^{(i)} \mid \rho \in Cone^{(i)}\}\,.$$

See Figure 2 for an example.

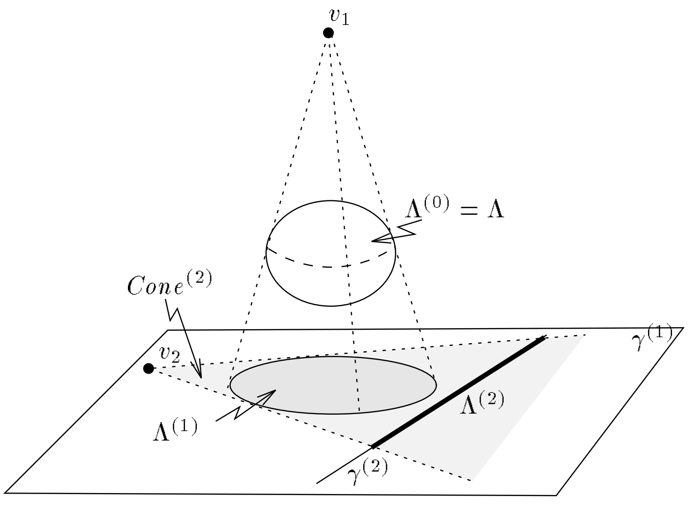


Figure 2: Projections of Λ.

Lemma 3.1 *For an oriented $(d-i-1)$-hyperplane h lying in $\gamma^{(i)}$,*

$$\operatorname{sign}_{\Lambda^{(i)}}(h) = \operatorname{sign}_{\Lambda}(\operatorname{aff}(h \cup V^{(i)}))\,.$$

Here we are assuming that the orientation of $\operatorname{aff}(h \cup V^{(i)})$ *is consistent with that of h (i.e., the projection of the positive normal of* $\operatorname{aff}(h \cup V^{(i)})$ *onto $\gamma^{(i)}$ is same as the positive normal of h.)*

Proof: The proof follows immediately by induction on i and the definition of $\Lambda^{(i)}$. $\square$

The i^{th} step of the algorithm either returns a hyperplane intersecting Λ or computes the vertex v_i and the hyperplane $\psi_i \in G$. This is done as follows. If $i = d+1$, then $\gamma^{(d)}$ is a point. We set v_{d+1} to be that point, and $\gamma^{(d+1)} = \emptyset$, and we are done. So assume that $i \leq d$. For the sake of convenience we regard $\gamma^{(i-1)}$ as $\mathbb{R}^{d-i+1}$. Let

$$G^{(i)} = \{h_j^{(i)} = h_j \cap \gamma^{(i-1)} \mid h_j \in G - \Psi^{(i-1)}\}$$

be a collection of oriented hyperplanes, where the orientation of $h_j^{(i)}$ is consistent with that of h_j. By our assumption on hyperplanes in G being in general position, $G^{(i)}$ consists of $t-i+1$ $(d-i)$-hyperplanes. (If the hyperplanes in G are not in general position, we include only those hyperplanes in $G^{(i)}$ that intersect $\gamma^{(i-1)}$ but do not contain $\gamma^{(i-1)}$, so in general $|G^{(i)}|$ consists of at most $t-i+1$ $(d-i)$-hyperplanes.) Let $b_j = \mathrm{aff}(h_j^{(i)} \cup V^{(i-1)})$ be the $(d-1)$-hyperplane passing through $h_j^{(i)}$ and $V^{(i-1)}$ and

$$B^{(i)} = \{b_j = \mathrm{aff}(h_j^{(i)} \cup V^{(i-1)}) \mid h_j^{(i)} \in G_j^{(i)}\}\,.$$

By Lemma 3.1, $\mathrm{sign}_{\Lambda^{(i-1)}}(h_j^{(i)}) = \mathrm{sign}_\Lambda(h_j^{(i)} \cup V^{(i-1)}) = \mathrm{sign}_\Lambda(b_j)$.

We compute $\mathrm{sign}_\Lambda(b_j)$ for all $b_j \in B^{(i)}$. If $\mathrm{sign}_\Lambda(b_j) = 0$, for some $b_j \in B^{(i)}$, we return b_j and stop, because we have found a hyperplane that intersects Λ. So assume that $\mathrm{sign}_\Lambda(b_j) \in \{+1, -1\}$ for all hyperplanes in $B^{(i)}$. Let $C^{(i)}$ be the cell of $\mathcal{A}(G^{(i)})$ corresponding to the sign sequence $(\mathrm{sign}_{\Lambda^{(i-1)}}(h_1^{(i)}), \ldots, \mathrm{sign}_{\Lambda^{(i-1)}}(h_{t-i+1}^{(i)}))$. We assume that $C^{(i)}$ is bounded; otherwise, we add a hyperplane at infinity to delimit $C^{(i)}$. We compute the topmost vertex (the maximum vertex in reverse lexicographical ordering) of $C^{(i)}$ by exhaustively searching all vertices of $\mathcal{A}(G^{(i)})$; this topmost vertex is the desired vertex v_i. Next, we compute the hyperplane ψ_i. If $\Lambda^{(i-1)}$ lies in a simplex of $CT(C^{(i)})$, then there is a facet g of $C^{(i)}$ such that all rays in $Cone^{(i)}$ intersect g before intersecting any other hyperplane of $G^{(i)}$. (Of course, the converse of this statement needs not be true.) We set $\psi_i = h_j$ if $h_j^{(i)}$ is the hyperplane supporting g. If there is no such facet of $C^{(i)}$, we conclude that $\Lambda^{(i-1)}$ is not contained in any simplex of $CT(C^{(i)})$, so that we can return a hyperplane intersecting Λ; namely the affine hull of $V^{(i-1)}$ and some facet of $CT(C^{(i)})$ that meets $\Lambda^{(i-1)}$. Next, we describe in detail how to find the hyperplane h_j.

Definition 3.2: Let ℓ_1 and ℓ_2 be two oriented $(k-1)$-hyperplanes in $\mathbb{R}^k$, and let the equation of ℓ_i be $a_1^i x_1 + \cdots + a_k^i x_k = b_i$. We say that $\ell_1 \underset{\text{rlex}}{<} \ell_2$ if $(a_1^1, \ldots, a_k^k, b_1) \underset{\text{rlex}}{<} (a_1^2, \ldots, a_k^2, b_2)$. Let ρ be a ray emanating from a point $p \notin \ell_1, \ell_2$. Let a_{12} be the hyperplane passing through $\ell_1 \cap \ell_2$ and p; ρ lies in one of the two half-spaces bounded by a_{12}, say in a_{12}^+. Let ℓ_i^+, for $i = 1, 2$, be the half-space bounded by ℓ_i that contains p. For a ray ρ as above not contained in the hyperplane a_{12}, we say $\ell_1 \underset{\rho}{<} \ell_2$ if ℓ_1 intersects the interior of the wedge formed by $\ell_2^+ \cap a_{12}^+$. In order to make $\underset{\rho}{<}$ a total ordering, for a ray $\rho \subseteq a_{12}$, we say $\ell_1 \underset{\rho}{<} \ell_2$ if $\ell_1 \underset{\text{rlex}}{<} \ell_2$.

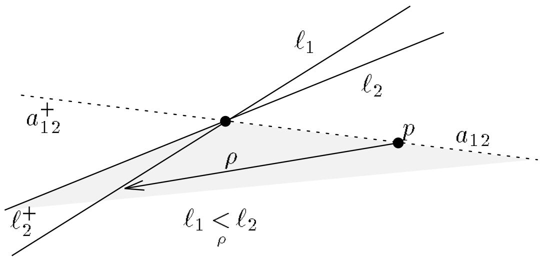


Figure 3: Illustration of the ordering induced by ρ.

It is easily seen that if ρ intersects ℓ_2 and $\ell_1 \underset{\rho}{<} \ell_2$, then ρ also intersects ℓ_1, and $d(p, \ell_1 \cap \rho) \leq d(p, \ell_2 \cap \rho)$. Moreover, $\ell_1 \underset{\rho}{<} \ell_2$ for all rays ρ emanating from p and contained in the (open) half-space a_{12}^+. Let $L = \{\ell_1, \ldots, \ell_m\}$ be a set of m hyperplanes in $\mathbb{R}^k$, and let ρ be a ray; $\underset{\rho}{<}$ defines a total ordering on L, and the first hyperplane of L intersected by ρ (assuming that such a hyperplane exists) is the minimum hyperplane in this ordering. Thus, by performing $m - 1$ comparisons, the first hyperplane intersected by ρ can be computed by using any minimum finding algorithm; see Figure 4.

Algorithm 1: FIND_MIN (L, ρ)
Input: $L = \{\ell_1, \ldots, \ell_m\}$, a set of $(k-1)$-hyperplanes in $\mathbb{R}^k$, and ρ, a ray emanating from a point p;
Output: The minimum hyperplane of L w.r.t. $\underset{\rho}{<}$;

```
ξ := ℓ_1;
for j = 2 to m do
    g := aff((ξ ∩ ℓ_j) ∪ {p});
    g+ := half-space bounded by g and containing ρ;
    ξ+ := half-space bounded by ξ and containing p;
    if (ρ ⊆ g and ℓ_j <_rlex ξ) or (ℓ_j ∩ int(ξ+ ∩ g+) ≠ ∅)
        then ξ := ℓ_j;
return ξ;
```

Figure 4: Algorithm for finding the first hyperplane intersected by a ray.

In order to compute ψ_i we will simulate the FIND_MIN algorithm, using Megiddo's parametric search technique [22], on the hyperplanes of $G^{(i)}$ that do not contain v_i, with a generic ray $\rho \in Cone^{(i)}$. At each step the FIND_MIN algorithm chooses two hyperplanes $h_j^{(i)}$ and $h_k^{(i)}$, one of which, say $h_j^{(i)}$, is the minimum of the hyperplanes compared so far. In order to determine whether $h_j^{(i)} \underset{\rho}{<} h_k^{(i)}$, we compute $h_{jk} = h_j^{(i)} \cap h_k^{(i)}$, let $a_{jk} = \mathrm{aff}(h_{jk} \cup \{v_i\})$ and $g_{jk} = \mathrm{aff}(a_{jk} \cup V^{(i-1)})$; a_{jk} is a $(d-i)$-hyperplane and g_{jk} is a $(d-1)$-hyperplane. By Lemma 3.1, $\mathrm{sign}_{\Lambda^{(i-1)}}(a_{jk}) = \mathrm{sign}_{\Lambda}(g_{jk})$, so we compute $\mathrm{sign}_{\Lambda}(g_{jk})$ by calling our oracle. If $\mathrm{sign}_{\Lambda}(g_{jk}) = 0$, we return g_{jk} and stop. Otherwise, $\Lambda^{(i-1)}$ lies in one of the two (open) half-spaces determined by a_{jk}, say a_{jk}^+. For all rays ρ emanating from v_i and contained in a_{jk}^+, $\min_\rho\{h_j^{(i)}, h_k^{(i)}\}$ is the same and does not depend anymore on the oracle. We set $\xi = h_k^{(i)}$ if $h_k^{(i)}$ intersects the interior of the wedge $h_j^{(i)+} \cap a_{ij}^+$, where $h_j^{(i)+}$ is the half-space bounded by $h_j^{(i)}$ and containing v_i. Thus, by calling the oracle once, we compute ξ at each step of the FIND_MIN algorithm. Hence, after invoking the oracle at most $|G^{(i)}|$ times, we either find a hyperplane intersecting Λ, or the desired hyperplane ψ_i.

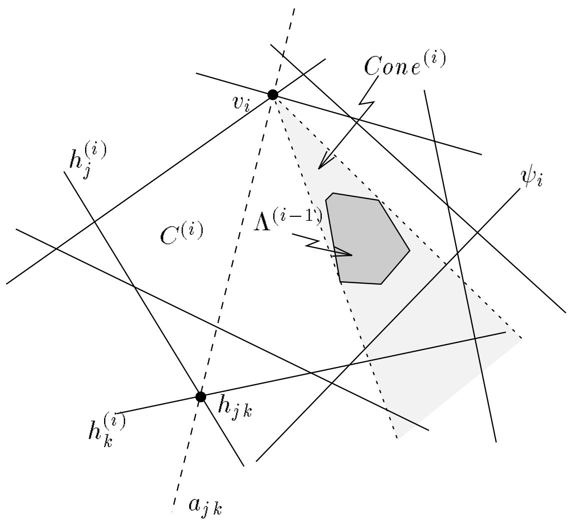


Figure 5: Illustration of simulating FIND_MIN .

This completes the description of the i^{th} step of the algorithm. We now prove the correctness of the algorithm. We first prove an auxiliary lemma.

Lemma 3.3 *If a hyperplane intersecting Λ has not been found by the algorithm, then a ray $\rho \in Cone^{(i)}$ either does not intersect any hyperplane of $G^{(i)}$, or intersects $\psi_i \cap \partial C^{(i)}$ before intersecting any hyperplane of $G^{(i)}$.*

Proof: We prove the lemma by induction on j, the number of comparisons made by the FIND_MIN algorithm. Let $H_j^{(i)} = \{h_1^{(i)}, \ldots, h_{j+1}^{(i)}\}$ be the subset of $j+1$ hyperplanes of $G^{(i)}$, which are compared by FIND_MIN in the first j steps. Let $C_j^{(i)}$ be the cell of $\mathcal{A}(H_j^{(i)})$

containing the cell $C^{(i)}$. We claim that if a hyperplane intersecting $\Lambda^{(i-1)}$ has not been found, then a ray $\rho \in Cone^{(i)}$ either does not intersect any hyperplane of $H_j^{(i)}$, or intersects $\xi \cap \partial C_j^{(i)}$ first, where ξ is the minimum hyperplane in $H_j^{(i)}$ (w.r.t. $\underset{\rho}{<}$).

The lemma is obviously true for $j = 0$, as this case involves only one hyperplane. Suppose the claim holds for all $l < j$, and consider the j^{th} step. Suppose $\xi = h_u^{(i)}$ after Find_Min has made its first $j-1$ comparisons. In the j^{th} step it compares $h_{j+1}^{(i)}$ with $h_u^{(i)}$. If $\mathrm{sign}_{\Lambda^{(i-1)}}(a_{(j+1)u}) = 0$, Find_Min returns the hyperplane $g_{(j+1)u}$. Otherwise this sign determines a unique side $a^+_{(j+1)u}$ of $a_{(j+1)u}$ that fully contains $\Lambda^{(i-1)}$. Observe that Find_Min sets $\xi := \min_\rho\{h_u^{(i)}, h_{j+1}^{(i)}\}$, where the minimum is taken over the rays emanating from v_i and lying in $a^+_{(j+1)u}$, and that $Cone^{(i)} \subseteq a^+_{(j+1)u}$. Thus $\xi = \min_\rho\{h_u^{(i)}, h_{j+1}^{(i)}\}$ also holds for any generic ray $\rho \in Cone^{(i)}$. By induction hypothesis, each ray $\rho \in Cone^{(i)}$ intersects $h_u^{(i)} \cap \partial C_{j-1}^{(i)}$ before intersecting any hyperplane of $H_{j-1}^{(i)}$. This and the definition of the order $\underset{\rho}{<}$ ensure that the first intersection point of ρ and $H_j^{(i)}$ (if any) lies on $\xi \cap \partial C_j^{(i)}$. This completes the proof of the lemma. □

Next we prove that the algorithm maintains the following invariant:

Lemma 3.4 *If the algorithm does not return a hyperplane intersecting Λ after the first i steps, then $\Lambda \subseteq CH(\gamma^{(i)} \cup V^{(i)})$.*

Proof: Suppose the algorithm does not return a hyperplane intersecting Λ in the first i steps. Then by Lemma 3.3, all rays $\rho \in Cone^{(i)}$ intersect $\psi_i \cap \partial C^{(i)}$ (since we assume that $C^{(i)}$ is bounded, ρ has to intersect some face of $C^{(i)}$), and therefore $\Lambda^{(i)} \subseteq \psi_i \cap \partial C^{(i)} \subseteq \gamma^{(i)}$. We will prove the following stronger claim by induction on i:

$$\Lambda \subseteq CH(\Lambda^{(i)} \cup V^{(i)}).$$

The claim is obviously true for $i = 0$ as $\Lambda \subseteq \mathbb{R}^d$. Suppose the lemma is true for $j < i$. It is easily seen that the definition of $\Lambda^{(i)}$ and the facts that v_i, ψ_i appear on $\partial C^{(i)}$ and $\Lambda^{(i-1)} \subseteq C^{(i)}$ imply that

$$\Lambda^{(i-1)} \subseteq CH(\{v_i\} \cup \Lambda^{(i)}).$$

Thus, by induction hypothesis

$$\begin{aligned}
\Lambda &\subseteq CH(\Lambda^{(i-1)} \cup V^{(i-1)}) \\
&\subseteq CH(CH(\{v_i\} \cup \Lambda^{(i)}) \cup V^{(i-1)}) \\
&= CH(\Lambda^{(i)} \cup V^{(i)})
\end{aligned}$$

This completes the proof of the lemma. □

If the algorithm does not return a hyperplane intersecting Λ, we have a set $V^{(d+1)}$ of $d+1$ vertices, and by Lemma 3.4, $\Lambda \subseteq CH(V^{(d+1)} \cup \gamma^{d+1})$. Since $\gamma^{(d+1)} = \emptyset$,

$$\Lambda \subseteq CH(V^{(d+1)}) = \triangle .$$

Let $H_{\triangle} \subseteq H$ be the set of hyperplanes intersecting the interior of $\triangle$, and let v be any vertex of $\triangle$. For all $h \in H - H_{\triangle}$, $\mathrm{sign}_{\Lambda}(h) = \mathrm{sign}_{\triangle}(h) = \mathrm{sign}_{\{v\}}(h)$; the last equality follows from the fact that all vertices of $\triangle$ lie on the same side of h. Therefore, we can compute $\mathrm{sign}_{\Lambda}(h)$ in $O(d)$ time without any additional oracle calls. Since G is a $(1/r)$-net for the range space H_d, each edge of $\triangle$ intersects at most n/r hyperplanes of H. Moreover, each hyperplane of $H_{\triangle}$ intersects at least one of the d edges incident to v. Hence, $|H_{\triangle}| \leq dn/r$. If we choose $r = 2d$, then the above algorithm resolves the signs of at least $n/2$ hyperplanes of H.

Next, we analyze the total number of oracle calls made by the algorithm. In the i^{th} step, it makes at most $t-i$ oracle calls to find the sign sequence of $C^{(i)}$, and at most $t-i-1$ additional oracle calls to find the hyperplane ψ_i. Thus the total number of oracle calls made over all steps is $2\sum_{i=1}^{d-1}(t-i) = O(d^2 r \log dr)$. Since $r = 2d$, the above algorithm resolves the signs of at least $n/2$ hyperplanes by making $O(d^3 \log d)$ oracle calls.

Finally, let us bound the time spent by the algorithm in doing other computations. We spend $n \cdot O(d)^{10d} \log^{2d} d$ time to compute the $(1/2d)$-net G. In the i^{th} step, we spend $O(d^3 \log d)$ time to compute $G^{(i)}$ and to simulate the FIND_MIN algorithm, and

$$O(d^2 \log d)^d \cdot d = O(d)^{3d}$$

time to compute the topmost vertex of $C^{(i)}$. After having computed $\triangle$, we spend $O(nd^2)$ time to compute the subset $H_{\triangle}$ and the numbers $\mathrm{sign}_{\Lambda}(h)$ for all $h \in H - H_{\triangle}$. Hence, the total time spent in doing other computations is $n \cdot O(d)^{10d} \log^{2d} d$.

Theorem 3.5 *Given a set H of n hyperplanes in $\mathbb{R}^d$ and an oracle $\mathcal{O}$ as defined in the Introduction, one can either compute a hyperplane intersecting Λ or resolve the signs of at least half of the hyperplanes in H by making $O(d^3 \log d)$ oracle calls. The time spent in doing other computations is $n \cdot O(d)^{10d} \log^{2d} d$.*

If we have a weight function $w : H \to \mathbb{R}^d$ associated with H, then using Corollary 2.3, we can compute a $(1/r)$-net for the weighted range space (H_d, w), and follow the same algorithm. The above discussion in conjunction with Corollary 2.3 implies that the algorithm resolves the signs of a subset $N \subseteq H$ such that $w(N) \geq w(H)/2$. Hence, we obtain

Theorem 3.6 *Given a set H of n hyperplanes, a weight function $w : H \to \mathbb{R}^+$, and an oracle $\mathcal{O}$ as defined in the Introduction, one can either compute a hyperplane intersecting Λ or resolve the signs of a subset $N \subseteq H$ such that $w(N) \geq w(H)/2$. The algorithm makes $O(d^3 \log d)$ oracle calls, and the time spent in doing other computations is $n \cdot O(d)^{10d} \log^{2d} d$.*

By applying Theorem 3.5 recursively to $H_\triangle$, we can resolve the signs of all hyperplanes in H. The recursion stops when $|H_\triangle| \leq 2t = O(d^2 \log d)$, in which case we compute the signs of all hyperplanes directly using the oracle. Hence, the total number of oracle calls is $O(d^3 \log d \log n)$, and the time spent in doing other computations is $n \cdot O(d)^{10d} \log^{2d} d$. We thus obtain

Theorem 3.7 *Given a set H of n hyperplanes in $\mathbb{R}^d$ and an oracle $\mathcal{O}$ as defined in the Introduction, one can either compute a hyperplane intersecting Λ or resolve the signs of all hyperplanes by making $O(d^3 \log d \log n)$ oracle calls. The time spent in doing other computations is $n \cdot O(d)^{10d} \log^{2d} d$.*

Remark 3.8: A similar approach has been used by Meiser [24] for point location in an arrangement of hyperplanes.

4 Applications

In this section we describe some of the applications of Theorem 3.5. It is beyond the scope of this paper to mention all the problems whose performance (the dependence of their time complexity on d) can be improved by applying our technique. Instead we briefly sketch two sample applications: (i) linear programming and related problems, and (ii) multi-dimensional parametric searching.

4.1 Linear programming and related problems

We first apply our technique to *linear programming*, which is defined as follows: Given a set Γ of n half-spaces in $\mathbb{R}^d$ (called *constraints*) and a direction vector $\mathbf{c}$, find a point $\mathbf{x} \in \cap\Gamma$ that minimizes $\mathbf{c} \cdot \mathbf{x}$. We assume that d is fixed. Although polynomial time algorithms for linear programming have been known for a while [17, 18], they are not strongly polynomial. The first strongly polynomial time algorithm (assuming that d is a constant) was given by Megiddo [23]. The running time of his original algorithm was $\alpha(d)n$, where $\alpha(d) = 2^{O(2^d)}$. The value of $\alpha(d)$ was improved by Clarkson [5] and by Dyer [11] to $O(3^{d^2})$. Using Theorem 3.5, we describe an algorithm with $\alpha(d) = O(d)^{10d} \log^{2d} d$, which is a significantly better than $O(3^{d^2})$. Chazelle and Matoušek have independently developed another deterministic algorithm with time complexity $\alpha(d) = O(d)^{7d(1+o(1))}$. In the past few years, several randomized algorithms have been developed for linear programming [6, 26]. Clarkson's algorithm runs in expected time $O(d^2 n + d^{d/2+O(1)} \cdot \log n)$. Recently Kalai [16] and Matoušek et al. [21] have presented randomized algorithms with expected running time $O(d^2 n + e^{O(\sqrt{d \ln d})})$.

It turns out that most of these linear programming algorithms can be extended to solve various so-called 'LP-type problems', including the problems of computing the Euclidean

1-center and the smallest enclosing ellipsoid for a set of n points in $\mathbb{R}^d$; see [4, 12, 21] for details. Our technique also yields algorithms for these problems with improved dependence on d, but we omit these applications in this paper.

We now briefly sketch how Theorem 3.5 leads to an improved algorithm for linear programming. We assume that the reader is familiar with Megiddo's algorithm. Let H be the set of hyperplanes in $\mathbb{R}^d$ bounding the hyperplanes of Γ. We define Λ to be the set

$$\Lambda = \{\mathbf{x} \mid \mathbf{x} \in \bigcap \Gamma \text{ and } \mathbf{c} \cdot \mathbf{x} \leq \mathbf{c} \cdot \mathbf{y},\ \forall \mathbf{y} \in \bigcap \Gamma\}.$$

That is, Λ is the (possibly empty) set of solutions of the given linear programming problem. If the solution is not degenerate, then Λ is a single point and lies on d of the hyperplanes of H. Megiddo [23] observed that the oracle $\mathcal{O}$, which computes $\operatorname{sign}_\Lambda(h)$ for a hyperplane h, can be implemented by solving at most three instances of linear programming in $\mathbb{R}^{d-1}$, each involving at most n constraints (see [23] for details). Moreover, $\mathcal{O}$ can also determine whether $\Lambda = \emptyset$, and, if $\Lambda = \emptyset$, $\mathcal{O}$ returns a certificate consisting of $d+1$ constraints whose intersection is empty. Henceforth we assume that $\Lambda \neq \emptyset$.

Theorem 3.5 implies that, by making $O(d^3 \log d)$ oracle calls, we either find a hyperplane π that intersects Λ, or resolve the signs of at least half of the hyperplanes in H. In the former case the solution of the original linear programming problem can be obtained by solving a $(d-1)$-dimensional linear programming problem with $\Gamma' = \{\gamma \cap \pi \mid \gamma \in \Gamma\}$ as the set of constraints and the orthogonal projection of $\mathbf{c}$ on π as the minimization direction vector. In the latter case let H' be the subset of hyperplanes whose signs have been resolved. If we find a hyperplane $h \in H'$ such that Λ does not lie in the half-space of Γ bounded by h, we conclude that there is no feasible solution, and stop right away. Otherwise, we discard the constraints corresponding to the hyperplanes in H', because Λ does not lie on any hyperplane in H', and we recursively solve the problem with the remaining constraints. The recursion stops when we are left with $O(d^2 \log d)$ constraints, in which case we use a brute-force method to find the solution. We have to consider at most

$$\binom{O(d^2 \log d)}{d} = O(d^2 \log d)^d$$

vertices, and we can determine the optimal solution by spending $O(d) \times O(d^2 \log d) = O(d^3 \log d)$ time at each of these vertices.

Let $LP(n, d)$ denote the maximum time spent by this algorithm in solving an instance of linear programming in $\mathbb{R}^d$ with n constraints. Then we obtain the following recurrence

$$LP(n,d) \leq \begin{cases} O(d)^{2d+3} \log^{d+1} d & \text{for } n \leq c_1 d^2 \log d\,, \\ LP(n/2, d) + c_2 d^3 \log d \cdot LP(n, d-1) & \\ +c_3 n \cdot (c_4 d)^{10d} \log^{2d} d & \text{for } n > c_1 d^2 \log d\,, \end{cases} \tag{4.1}$$

where c_1, c_2, c_3, c_4 are some appropriate absolute constants. (The constant c_2 is 3 times the constant of proportionality in the bound on the number of oracle calls, and is easily seen to be reasonably small.) The solution of the above recurrence is

$$LP(n,d) \le An \cdot (c_4 d)^{10d} (c_2 \log^2 d)^d$$

for some sufficiently large absolute constant $A > 0$. Indeed, the claim is obviously true for $n \le c_1 d^2 \log d$. For larger values of n, by induction on n, we obtain

$$\begin{aligned}
LP(n,d) &\le LP(n/2,d) + c_2 d^3 \log d \cdot LP(n,d-1) + c_3 n \cdot (c_4 d)^{10d} \log^{2d} d \\
&\le A\frac{n}{2} \cdot (c_4 d)^{10d} (c_2 \log^2 d)^d + c_2 d^3 \log d \cdot An \cdot (c_4(d-1))^{10(d-1)} (c_2 \log^2 (d-1))^{d-1} \\
&\quad + c_3 n \cdot (c_4 d)^{10d} \log^{2d} d \\
&\le An \cdot (c_4 d)^{10d} (c_2 \log^2 d)^d \left(\frac{1}{2} + \frac{c_2}{c_4 d^7 \log d} + \frac{c_3}{A} \right) \\
&\le An \cdot (c_4 d)^{10d} (c_2 \log^2 d)^d ,
\end{aligned}$$

provided that A is chosen sufficiently large. Hence, we can conclude

Theorem 4.1 *An instance of linear programming in $\mathbb{R}^d$ with n constraints can be solved in time $n \cdot O(d)^{10d} \log^{2d} d$.*

Remark 4.2: (i) The bottleneck in the time complexity of Theorem 4.1 is computing a $(1/2d)$-net. The time spent in doing other computations is only $n \cdot O(d)^{3d} \log^{2d} d$.

(ii) Notice that the time compelxity of our algorithm is slightly worse than that of the algorithm due to Chazelle and Matoušek [4]. The dominating factor in the time complexity of their algorithm is also computing a $(1/r)$-net. However, their algorithm is tuned to solving LP-type problems, so it needs to compute a $(1/r)$-net for the range space

$$\Sigma_d = (H, \{\, \{ h \in H \mid h \cap \rho \neq \emptyset \} \mid \rho \text{ is vertical ray in } +x_d\text{-direction} \,\})$$

whose VC-dimension is only d, which plugging into Theorem 2.1 yields the time complexity of their algorithm.

4.2 Multi-dimensional parametric search

In this subsection we apply Theorem 3.5 to the multi-dimensional parametric search technique for maximizing a concave function. Let $Q \subseteq \mathbb{R}^d$ and let $F : Q \to \mathbb{R}$ be a piecewise linear concave function. Let $\Lambda \subseteq Q$ be the set of points at which F is maximized; Λ is called the *maximizers* of F. Let $\mathcal{A}_p$ be a parallel algorithm (in Valiant's weak comparison model [27]) that takes as input $\mathbf{x} \in \mathbb{R}^d$ and returns $F(\mathbf{x})$ if $\mathbf{x} \in Q$. For $\mathbf{x} \notin Q$ $\mathcal{A}_p$ stops and returns an arbitrary value. An algorithm is said to be *piecewise affine* if the only operations performed on intermediate values that depend on $\mathbf{x}$ are additions, multiplications by constants, and comparisons. Megiddo and Cohen [7, 8, 9] and independently Norton, Plotkin and Tardos [25] showed that if $\mathcal{A}_p$ is piecewise affine and runs in ζ parallel steps using π processors, then a maximizer of F can be found in $O(3^{d^2} \cdot (\zeta \log \pi)^d \zeta\pi)$ time. We outline their technique here, and refer the reader to the original papers for details.

As in Megiddo's linear programming algorithm, given a hyperplane h in $\mathbb{R}^d$, $\text{sign}_\Lambda(h)$ can be determined by at most 3 calls to the maximization algorithm in $\mathbb{R}^{d-1}$ on problems of the same input size (this follows from the concavity of F). The general technique is an extension of Megiddo's parametric searching technique [22]. The algorithm $\mathcal{A}_p$ is simulated on a maximizer $\mathbf{x}^\star \in \Lambda$, which is carried through the algorithm as an unknown symbolic value. The arithmetic operations are performed on affine expressions. Consider a set of comparisons that must be resolved in some step of the execution. Each comparison is made between two affine expressions, and is therefore equivalent to computing $\text{sign}_\Lambda(h)$ for an appropriate hyperplane h; h is called a *critical hyperplane*. At each parallel step, $\mathcal{A}_p$ performs $t \leq \pi$ comparisons, therefore one has to compute the sign of t hyperplanes $h_1, \ldots, h_t$. By Theorem 3.5, this can be done by making $O(d^3 \log d \log \pi)$ oracle calls and by spending additional $\pi \cdot O(d)^{10d} \log^{2d} d$ time in doing auxiliary computations. If a hyperplane h intersecting Λ is returned, then $\max_{\mathbf{x} \in Q} F(\mathbf{x}) = \max_{\mathbf{x} \in Q \cap h} F(\mathbf{x})$, so the problem has been reduced to finding a maximizer on h. If $d = 1$, then h is a point and we can compute $F(h)$ by invoking $\mathcal{A}_p$ on h. The total time spent is obviously at most $O(\zeta \cdot \pi)$. For $d > 1$, we can regard h as $\mathbb{R}^{d-1}$, so the problem can be solved by simulating $\mathcal{A}_p$ at a point in $\Lambda \cap h$. On the other hand, if no hyperplane intersecting Λ is found, the signs of all critical hyperplanes are determined, and thus all the comparisons of the current parallel step can be resolved, so the simulation of $\mathcal{A}_p$ can be continued. If $\text{sign}(h_i) = +1$ (resp. $\text{sign}(h_i) = -1$), then the half-space $h_i \geq 0$ (resp. $h_i \leq 0$) acts as a *constraint* for Λ, i.e., Λ is contained in that half-space. If the simulated algorithm $\mathcal{A}_p$ terminates without finding a hyperplane intersecting Λ, it returns a (symbolic) value which depends on the unknown $\mathbf{x}^*$. It can be shown that the value returned is an affine function $g : \mathbb{R}^d \to \mathbb{R}$ of $\mathbf{x}^*$. Let Γ be the set of constraints for Λ. It then follows that a maximizer for F can be found by solving the following linear programming problem: find a point $\mathbf{x} \in \bigcap \Gamma$ that maximizes $g(\mathbf{x})$ (which is equivalent to minimizing $-g(\mathbf{x})$). Since $|\Gamma| \leq \zeta \cdot \pi$, such a point $\mathbf{x}$ can be computed in time $LP(\zeta \cdot \pi, d) = \zeta\pi \cdot O(d)^{10d} \log^{2d} d$.

Let $T(d)$ denote the maximum running time of this parametric searching technique in $\mathbb{R}^d$, then we obtain the following recurrence:

$$T(d) \leq \begin{cases} \zeta \cdot \pi & \text{for } d = 0, \\ LP(\zeta\pi, d) + \zeta\pi \cdot O(d)^{10d} \log^{2d} d + \zeta \log \pi \cdot c d^3 \log d \cdot T(d-1) & \text{for } d > 0. \end{cases}$$

where c is some constant. It can be checked, in a manner similar to the analysis of the preceding subsection, that the solution of the recurrence is

$$T(d) \leq O(d)^{10d} \log^{2d} d \cdot (\zeta \log \pi)^d \zeta \pi \ .$$

This is an improvement over the use of the multi-dimensional searching techniques of Dyer [11] and Clarkson [5], which yield a factor of $O(3^{d^2})$ in the running time. A similar improvement in the dependence on d can be obtained for the recent convex maximization algorithm of Agarwala and Fernández-Baca [1].

References

[1] R. Agarwala and D. Fernández-Baca, Weighted multidimensional search and its application to convex optimization, Tech. Report 92–51, DIMACS, Rutgers University, 1992.

[2] B. Chazelle, An optimal convex hull algorithm and new results on cuttings, *Proc. 32nd IEEE Annual Symp. on Foundations of Computer Science*, 1991, 29–38.

[3] B. Chazelle and J. Friedman, A deterministic view of random sampling and its use in geometry, *Combinatorica*, **10** (1990), 229–249.

[4] B. Chazelle and J. Matoušek, On linear-time deterministic algorithms for optimization problems in fixed dimension, *Proc. 4th ACM-SIAM Symp. on Discrete Algorithms*, 1993, 281–290.

[5] K.L. Clarkson, Linear programming in $O(n3^{d^2})$ time, *Inf. Proc. Lett.*, **22** (1986), 21–24.

[6] K. Clarkson, Las Vegas algorithm for linear programming when the dimension is small, *Proc. 29th IEEE Symposium on Foundations of Computer Science*, 1988, pp. 452–457.

[7] E. Cohen and N. Megiddo, Strongly polynomial-time and NC algorithms for detecting cycles in dynamic graphs, *Proc. 21st Annual ACM Symp. on Theory of Computing* 1989, 523–534.

[8] E. Cohen and N. Megiddo, Algorithms and complexity analysis for some flow problems, *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, 1991, 120–130.

[9] E. Cohen and N. Megiddo, Maximizing concave functions in fixed dimension, in *Complexity in Numerical Optimization*, P.M. Pardalos, ed., World Scientific, 1993, 75–87. (See also Research Report RJ 7656 (71103), IBM Almaden Research Center, San Jose, 1990.)

[10] R. Cole, Slowing down sorting networks to obtain faster sorting algorithms, *J. Assoc. Comput. Mach.*, **34** (1987), 200–208.

[11] M.E. Dyer, On a multidimensional search technique and its application to the Euclidean one-centre problem, *SIAM J. Comput.*, **15** (1986), 725–738.

[12] M. E. Dyer, A class of convex programs with applications to computational geometry, *Proc. 8th ACM Symposium on Computational Geometry*, 1992, 9–15.

[13] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, 1987, Springer–Verlag, New York–Berlin.

[14] D. Fernández-Baca and G. Slutzki, Parametric problems on graphs of bounded tree-width, *Proc. 3rd Scandinavian Workshop on Algorithm Theory*, Lecture Notes in Computer Science 621, 1992, pp. 304–316.

[15] D. Haussler and E. Welzl, ϵ-nets and simplex range queries, *Discrete & Computational Geometry*, **2** (1987), 127–151.

[16] G. Kalai, A subexponential randomized simplex algorithm. *Proc. 24th ACM Symposium on Theory of Computing*, 1992, 475–482.

[17] N. Karmakar, A new polynomial-time algorithm for linear programming, *Combinatorica*, 4 (1984), 373–395.

[18] L. Khachiyan, Polynomial algorithm in linear programming, *U.S.S.R. Comput. Math. and Math. Phys.*, **20** (1980), 53–72.

[19] J. Matoušek, More on cutting hyperplanes and spanning trees with low crossing number, Tech. Rept., Freie Universität Berlin, 1990.

[20] J. Matoušek, Approximations and optimal geometric divide-and-conquer, *Proc. 23rd ACM Symposium on Theory of Computing*, 1991, 506–511.

[21] J. Matoušek, M. Sharir, and E. Welzl, A subexponential bound for linear programming, *Proc. 8th ACM Symposium on Computational Geometry*, 1992, 1–8.

[22] N. Megiddo, Applying parallel computation in the design of serial algorithms, *J. Assoc. Comput. Mach.*, **30** (1983), 852–865.

[23] N. Megiddo, Linear programming in linear time when the dimension is fixed, *J. Assoc. Comput. Mach.*, **31** (1984), 114–127.

[24] S. Meiser, Point location in arrangements, *Computational Geometry and its Applications*, Lecture Notes in Computer Science, vol. 333, 71–84.

[25] C. Norton, S. Plotkin and É. Tardos, Using separation algorithms in fixed dimension, *J. Algorithms*, **13** (1992), 79–98.

[26] R. Seidel, Small dimensional linear programming and convex hulls made easy, *Discrete & Computational Geometry*, **6** (1991), 423–434.

[27] L. Valiant, Parallelism in comparison problems, *SIAM J. Comput.*, **4** (1975), 345-348.