Deterministic Algorithms for 2-d Convex Programming and 3-d Online Linear Programming*

Timothy M. Chan

Department of Mathematics and Computer Science
University of Miami, Coral Gables, FL 33124–4250, USA
E-mail: tchan@cs.miami.edu

October 31, 1997

Abstract

We present a deterministic algorithm for solving two-dimensional convex programs with a linear objective function. The algorithm requires $O(k \log k)$ primitive operations for k constraints; if a feasible point is given, the bound reduces to $O(k \log k / \log\log k)$. As a consequence, we can decide whether k convex n-gons in the plane have a common intersection in $O(k \log n \min\{\log k, \log\log n\})$ worst-case time. Furthermore, we can solve the three-dimensional online linear programming problem in $o(\log^3 n)$ worst-case time per operation.

Running Head: 2-d Convex Programming

1 Introduction

Convex programming in fixed-dimensional space is a fundamental problem in computational geometry with many applications. Using randomization, the problem has been solved satisfactorily, as simple methods are known that require only a linear expected number of operations. These methods include a random sampling algorithm by Clarkson [8] and a randomized incremental algorithm by Sharir and Welzl [28]; the latter one is based on a linear programming algorithm of Seidel [27]. These methods are not limited to convex programming but in fact apply to a general class of *LP-type problems* [28].

Despite the success of randomized methods to the convex programming problem, deterministic methods lack behind both in terms of simplicity and generality. There are currently several deterministic approaches to the problem. One approach is to extend the *prune-and-search* algorithms for linear programming due to Megiddo and Dyer [13, 19, 20]. Dyer [14] reported linear-time algorithms for a restricted class of convex programs in which only a few of the constraints may be non-linear. Extension to the general case of convex programming is not known at the moment.

*A portion of this work was performed while the author was visiting the Center for Geometric Computing at Johns Hopkins University, and it was supported in part by the Army Research Office under grant DAAH04-96-1-0013. A preliminary version of this paper appeared in *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 464–472, 1997.

A more general deterministic approach [29] is to apply the *parametric search* paradigm of Megiddo [18]. This approach is complicated and does not usually lead to linear-time algorithms. Furthermore, it requires each constraint to be defined by a constant number of polynomials of a fixed degree; thus, it does not work for arbitrary convex constraints.

The recent technique of *derandomization* has led to a new deterministic approach to convex programming. By derandomizing Clarkson's algorithm, Chazelle and Matoušek [7] showed that a subclass of LP-type problems can be solved in a linear number of operations. Most convex programs involving fixed-degree polynomials fall into this subclass, but as pointed out in their paper, there are convex programming problems of dimension 2 for which their method does not apply.

In this paper, we consider the two-dimensional convex programming problem with a linear objective function:

> Given a collection of k compact convex sets in the plane (the *constraints*) and a linear function (the *objective*), find a point that minimizes the function over the intersection of the k constraints.

Unlike the previous deterministic approaches, we do not impose any restriction on the constraints; we only assume that there are procedures for performing the following *primitive operations*: (i) optimize a linear function over the intersection of two constraints, and (ii) intersect a constraint with a line. The known randomized methods can solve the problem using an $O(k)$ expected number of such primitive operations. By examining solutions defined by all pairs of constraints, one can easily obtain a deterministic algorithm that uses $O(k^2)$ number of primitive operations. Using a prune-and-search technique, we give a faster deterministic algorithm that requires an $O(k \log k)$ number of primitive operations. Somewhat surprisingly, we are able to improve this number to $O(k \log k / \log\log k)$ if an initial feasible point (i.e., a point in the intersection) is given.

Our motivation for considering two-dimensional convex programs with general constraints stems from the following problem:

> Given a collection of k convex n-gons in the plane, decide whether their common intersection is empty.

In 1988, Reichling [24] used the prune-and-search paradigm to obtain an algorithm that runs in $O(k \log^2 n)$ worst-case time. Since then, no improvements were reported for this seemingly basic problem.

By viewing the n-gons as constraints, we can think of this problem as a convex program in which the primitive operations can be implemented in $O(\log n)$ time. Using known randomized methods for convex programming, one can then get an algorithm with an $O(k \log n)$ expected running time. Our deterministic method immediately yields an algorithm with an $O(k \log k \log n)$ worst-case running time. By refining our prune-and-search technique, we achieve an $O(k \log n \log\log n)$ worst-case time bound, which is a strict improvement over the result by Reichling. Whether the remaining gap between the randomized and deterministic complexities can be closed is left as an open problem.

As a further application of our technique, we consider the *online linear programming problem* in three dimensions:

Maintain a set of at most n halfspaces in $\mathbb{R}^3$ under a sequence of insertions such that one can quickly compute the minimum of any linear objective function over the intersection of this set of halfspaces.

The quality of a solution is measured in terms of the insertion time (the time for inserting a new halfspace) and the query time (the time for computing the optimum of a given objective function). The objective may change for different queries. Under the assumption that all sequences of insertions are equally likely, there is a randomized algorithm that achieves an $O(\log n)$ expected amortized insertion time and an $O(\log^2 n)$ query time with high probability; see Mulmuley's book [21, Section 3.2.3]. If in addition the objective function is fixed, then Seidel's algorithm [27] achieves constant expected amortized insertion time. For an arbitrary sequence of insertions, Eppstein [15] gave a randomized method with $O(\log n)$ insertion time and $O(\log^{7/2} n \log\log n)$ expected query time, as well as a deterministic method with $O(\log n)$ insertion time and $O(\log^6 n)$ query time.

With the known randomized methods for convex programming, it is possible to improve the expected query time bound of Eppstein's randomized method (this is noted at the end of his paper [15]). Here, we are interested in improving the $O(\log^6 n)$ query time bound of his deterministic method. By applying our technique for convex programming, we show how to obtain $o(\log^3 n)$ worst-case query and insertion time.

As in Eppstein's method, we can also permit deletions of halfspaces, if during each insertion of a halfspace, we are told the time at which the halfspace will be deleted. For the fully dynamic case though, the best results currently known are obtained from Matoušek's linear optimization techniques [17] (see also [4]) combined with data structures by Agarwal and Matoušek [1].

The remainder of this paper is organized as follows. In the next section, we present our convex programming algorithm in two dimensions. In Section 3, we consider the case where the constraints are convex n-gons. Section 4 discusses the application to the online linear programming problem in three dimensions. Finally, Section 5 mentions some remaining open problems.

2 Convex Programming in $\mathbb{R}^2$

In this section, we study the following problem: given a collection $\mathcal{C}$ of k compact convex sets in $\mathbb{R}^2$ and $a \in \mathbb{R}^2$, find a point $v \in \bigcap_{C\in\mathcal{C}} C$ minimizing $a \cdot v$. The known randomized methods for convex programming [8, 28] can solve the problem using $O(k)$ expected number of the following primitive operations:

(R1) Given $C, C' \in \mathcal{C}$, find point $v \in C \cap C'$ minimizing $a \cdot v$.

(R2) Given $C \in \mathcal{C}$ and point p, decide whether $p \in C$.

If a certain randomized algorithm of Clarkson [8] is used, the expected number of operation (R1) is only $O(\log k)$.

We will give deterministic algorithms that solve the problem using $O(k \log k)$ number of the following, slightly stronger, primitive operations:

(D1) Given $C, C' \in \mathcal{C}$, find points $v_{\min}, v_{\max} \in C \cap C'$ minimizing $a \cdot v_{\min}$ and $-a \cdot v_{\max}$.

(D2) Given $C \in \mathcal{C}$ and line ℓ, compute the line segment $C \cap \ell$.

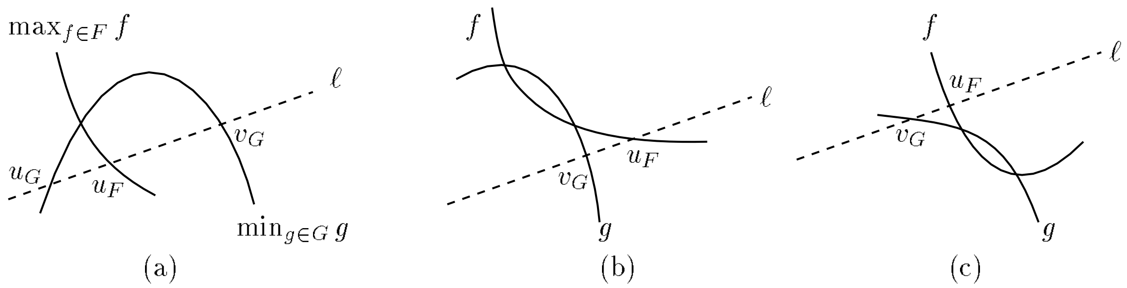

Figure 1: Simple tests determine whether v^* is above or below ℓ. (a) Case 1. (b,c) Case 2.

2.1 The algorithm

Without loss of generality, we may assume that $a = (1, 0)$, i.e., we want to find the leftmost point in $\bigcap_{C \in \mathcal{C}} C$. We assume that the problem is feasible; see Section 2.3 for how to handle the infeasible case. For simplicity, we ignore degenerate cases and assume that the optimal point, denoted by $v^* \in \mathbb{R}^2$, is unique; our algorithms can be modified to deal with degeneracy directly.

We first restate the problem in terms of univariate functions. For each convex set $C \in \mathcal{C}$, we first compute the x-coordinates $x_{\min}, x_{\max}$ of its leftmost and rightmost points, by performing an operation (D1). Define a convex function $f : \mathbb{R} \to \mathbb{R}$ such that within $[x_{\min}, x_{\max}]$, the graph of f coincides with the lower envelope of C; within $(-\infty, x_{\min})$, f is linear with slope approaching $-\infty$; and within $(x_{\max}, \infty)$, f is linear with slope approaching ∞. Similarly, define a concave function $g : \mathbb{R} \to \mathbb{R}$ corresponding to the upper envelope of C. Then the region bounded by the graphs of f and g is precisely C. The convex programming problem is:

> Given a collection F of k convex functions and a collection G of k concave functions on $\mathbb{R}$, find the left intersection point v^* of $\max_{f \in F} f$ and $\min_{g \in G} g$.

Note that in general two convex functions or two concave functions may intersect an arbitrary number of times. We observe, however, that a convex function and a concave function may intersect only twice (assuming non-degeneracy). Our algorithm uses the following as primitive operations:

(D1′) Given $f \in F$ and $g \in G$, find the two intersection points of f and g.

(D2′) Given $\phi \in F \cup G$ and line ℓ, find the at most two intersection points of ϕ and ℓ.

For the functions we have defined, operation (D1′) reduces to (D1) plus a constant number of (D2)'s, and operation (D2′) reduces to (D2).

We adopt a prune-and-search approach to compute v^*: in linear time, we identify a fraction of functions in F or a fraction of functions in G that cannot define v^*; then we eliminate these functions and repeat. Before we describe the algorithm, we first borrow some ideas used in previous prune-and-search algorithms of Dyer [13] and Megiddo [19] for two- and three-dimensional linear programming; these are encapsulated in the two lemmas below (proofs are included for completeness sake). In what follows, an *oracle* is a procedure that, given a line ℓ, decides which side of ℓ is v^* on, and computes v^* if v^* lies precisely on ℓ.

Lemma 2.1 *An oracle can be implemented using one operation* (D1) *and a linear number of operations* (D2).

Proof: We will only prove the lemma when ℓ is non-vertical; the vertical case is similar. Using operation (D2$'$), compute the left and right intersection points, u_F and v_F, of $\max_{f\in F} f$ with ℓ, and the left and right intersection points, u_G and v_G, of $\min_{g\in G} g$ with ℓ. Note that if $\max_{f\in F} f$ or $\min_{g\in G} g$ does not intersect ℓ, we can immediately conclude that v^* is strictly above or below ℓ.

Without loss of generality, suppose u_F is to the right of u_G. The oracle can be decided by some local tests (see Figure 1):

Case 1. u_F is to the left of v_G. If $u_F \neq u_G$, then v^* is strictly above ℓ. Otherwise, $v^* = u_F = u_G$.

Case 2. u_F is strictly to the right of v_G. Suppose u_F and v_G are defined by the functions $f \in F$ and $g \in G$. Intersect f and g via a single operation (D1$'$), and note that v^* is contained in the region bounded by f and g, which is either completely above or completely below ℓ. □

Lemma 2.2 *Given a set L of n non-vertical lines, we can accomplish one of the following by performing two oracle calls plus $O(n)$ additional work: (i) determine the position of v^*, or (ii) identify $\lceil n/8 \rceil$ lines $\ell \in L$ such that v^* lies strictly above ℓ, or (iii) identify $\lceil n/8 \rceil$ lines $\ell \in L$ such that v^* lies strictly below ℓ.*

Proof: Let m be the median slope of L. Form a set S of at least $n/2$ points as follows. Arbitrarily pair lines of slope $< m$ with lines of slope $> m$ and add the intersection of each pair of lines to S. Furthermore, for each line of slope exactly m, pick an arbitrary point on the line and add it to S.

Draw a vertical line through each point of S and consult the oracle for the median such line ℓ_1. Without loss of generality, say v^* is strictly to the left of ℓ_1. There are at least $n/4$ points of S to the right of ℓ_1. Draw lines of slope m through each such point and consult the oracle for the median such line ℓ_2. Without loss of generality, say v^* is strictly above ℓ_2.

Now, there are at least $n/8$ intersection points that are to the right of ℓ_1 and below ℓ_2. There are at least $n/8$ lines of slope $\geq m$ passing through these points. It is easy to see that v^* lies strictly above each of these lines. The time complexity of the whole procedure is $O(n)$, if we use a linear-time median-finding algorithm. □

Theorem 2.3 *A convex program in $\mathbb{R}^2$ with k constraints and a linear objective function can be solved using $O(k \log k)$ primitive operations.*

Proof: Let $p = |F|$ and $q = |G|$; initially, $p = q = k$. Without loss of generality, assume $p \geq q$; otherwise, swap F and G. Following Dyer's and Megiddo's linear programming algorithms, we pair functions and show how to remove one function out of each pair for a fraction of the pairs. However, there is a subtle difference in our approach that explains why we do not get the usual linear complexity in prune-and-search: our pairing is not arbitrary. Specifically, we only pair a convex function with a concave function. Since the number p of convex functions and the number q of concave functions may be different, our pairing may not be one-to-one. But it is possible to construct a pairing such that each convex function in F is paired with exactly one concave function in G, and each concave function in G is paired with at most $\lceil p/q \rceil$ convex functions in F.

For each pair (f, g) with $f \in F$ and $g \in G$, compute the two intersection points of f and g via operation (D1$'$) and construct the line ℓ passing through these two points. Observe that if v^* lies

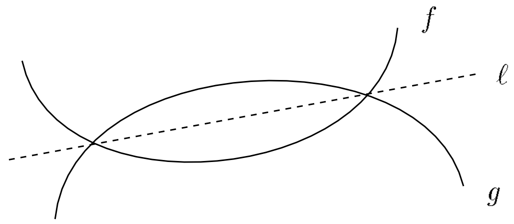

Figure 2: Prune f if v^* is strictly above ℓ. Prune g if v^* is strictly below ℓ.

strictly above ℓ, then f cannot define v^* and can thus be pruned. Similarly, if v^* lies strictly below ℓ, then g cannot define v^* and can be pruned. See Figure 2.

Let L be the set of p lines constructed from the p pairs. Invoke Lemma 2.2 on L. If case (i) of the lemma is reached, then we are done and the algorithm can be terminated. If case (ii) of the lemma is reached, then we can prune f from F for $\lceil p/8 \rceil$ of the pairs (f, g); as a consequence, the size of F reduces to at most $p - p/8 = 7p/8$. If case (iii) of the lemma is reached, then we can prune g from G for $\lceil p/8 \rceil$ of the pairs (f, g); since each g is paired with at most $\lceil p/q \rceil < 2p/q$ functions, the size of G reduces to at most $q - (p/8)/(2p/q) = 15q/16$. The process is now repeated for the new sets of functions F and G.

By Lemma 2.1, the oracle can be implemented in $O(Ap)$ time, where A denotes the time to perform a primitive operation. We thus have the following recurrence for the running time $T(p, q)$ for p convex functions and q concave functions:

$$T(p,q) \leq \begin{cases} \max\{T(\left\lfloor \frac{7}{8}p \right\rfloor, q), T(p, \left\lfloor \frac{15}{16}q \right\rfloor)\} + O(Ap) & \text{if } p \geq q \\ T(q,p) & \text{if } q > p \end{cases}$$

Note that $T(p, 0) = T(0, q) = O(1)$, as the solution is unbounded if either F or G is empty. The number of iterations is $O(\log p + \log q)$, hence $T(k, k) = O(Ak \log k)$. □

2.2 Refinement

In certain applications, operation (D1) is more costly than operation (D2). In this subsection, we note a modification to the algorithm of Theorem 2.3 that allows us to reduce the number of operations (D1) from $O(k \log k)$ to $O(k)$.

Theorem 2.4 *A convex program in* $\mathbb{R}^2$ *with* k *constraints and a linear objective function can be solved using* $O(k)$ *operations* (D1) *and* $O(k \log k)$ *operations* (D2).

Proof: We follow the algorithm of Theorem 2.3, except that we form fewer pairs at each iteration so as to reduce the number of operations (D1′). Let A_1 be the time needed for an operation (D1′) and let A_2 be the time needed for an operation (D2′). As before, let $p = |F|$ and $q = |G|$ with $p \geq q$. Choose a number r with $p \geq r \geq q$. Select r convex functions from F and apply our pairing strategy on these r convex functions with the q concave functions of G. This allows us to eliminate either $r/8$ functions from F or $q/16$ functions from G. The cost of such an iteration is $O(A_1 r + A_2 p)$, since we perform r operations (D1′) for the r pairs (note that an oracle requires only a constant number of operations (D1′) by Lemma 2.1).

It remains to describe the choice of the number r ($p \geq r \geq q$) for each iteration. Fix a number R. We consider three cases: (i) $p \geq q \geq R$, (ii) $p \geq R > q$, and (iii) $R > p \geq q$. We set $r = q$ in case (i), $r = R$ in case (ii), and $r = p$ in case (iii). This yields the following recurrence for the running time:

$$T(p,q) \leq \begin{cases} \max\{T(\lfloor p - \frac{q}{8} \rfloor, q),\, T(p, \lfloor \frac{15}{16} q \rfloor)\} \,+\, O(A_1 q + A_2 p) & \text{if } p \geq q \geq R \\ \max\{T(\left\lfloor p - \frac{R}{8} \right\rfloor, q),\, T(p, \left\lfloor \frac{15}{16} q \right\rfloor)\} \,+\, O(A_1 R + A_2 p) & \text{if } p \geq R > q \\ \max\{T(\left\lfloor \frac{7}{8} p \right\rfloor, q),\, T(p, \left\lfloor \frac{15}{16} q \right\rfloor)\} \,+\, O(A_1 p + A_2 p) & \text{if } R > p \geq q \\ T(q,p) & \text{if } q > p \end{cases}$$

It is a straightforward exercise to show by induction that

$$T(p,q) \quad = \quad O(A_1(p + q + R \log p + R \log q) \;+\; A_2(p \log q + q \log p + p^2/R + q^2/R)).$$

Setting $R = k/\log k$ implies that $T(k,k) = O(A_1 k + A_2 k \log k)$. □

2.3 Dealing with infeasibility

In the previous subsections, we have assumed that the given convex program is feasible. Suppose this is not the case. Then the point v^* generated by our algorithms cannot be feasible. Thus, we can detect whether a given convex program is feasible by simply testing the feasibility of this point (which requires a linear number of primitive operations). In other words, we can use the algorithms in Theorems 2.3 and 2.4 to determine whether $\bigcap_{C \in \mathcal{C}} C = \emptyset$ for a given collection $\mathcal{C}$ of convex sets in $\mathbb{R}^2$.

In certain applications, knowing that $\bigcap_{C \in \mathcal{C}} C = \emptyset$ is not enough; we may also want a *witness*—a triplet $C_1, C_2, C_3 \in \mathcal{C}$ satisfying $C_1 \cap C_2 \cap C_3 = \emptyset$. The existence of a witness is guaranteed by Helly's theorem. It is possible to modify our algorithms directly so that a witness is produced whenever the convex program is infeasible. Here, we point out instead a general approach. (Extension to an arbitrary fixed dimension is obvious.)

Observation 2.5 *Suppose we have an algorithm that decides whether the intersection of k convex sets in $\mathbb{R}^2$ is empty in $T_0(k)$ time (assuming $T_0(k)/k$ is a non-decreasing function). Given a collection $\mathcal{C}$ of k convex sets in $\mathbb{R}^2$ whose intersection is empty, a witness for $\mathcal{C}$ can be found in $O(T_0(k))$ time.*

Proof: Partition $\mathcal{C}$ into four subcollections $\mathcal{C}_1, \ldots, \mathcal{C}_4$, each containing at least $\lfloor k/4 \rfloor$ convex sets. Let $\mathcal{C}_i' = \mathcal{C} - \mathcal{C}_i$ ($i = 1, \ldots, 4$). By Helly's theorem, at least one of the collections $\mathcal{C}_i'$ has an empty intersection, and such a $\mathcal{C}_i'$ can be determined in $4T_0(\lceil 3k/4 \rceil)$ time. We recursively find a witness for this collection $\mathcal{C}_i'$ of at most $\lceil 3k/4 \rceil$ convex sets, until only three sets remain. The running time satisfies the recurrence $T(k) \leq T(\lceil 3k/4 \rceil) + O(T_0(\lceil 3k/4 \rceil))$, which solves to $T(k) = O(T_0(k))$. □

2.4 When a feasible point is given

In this subsection, we show how to improve the time bound in Theorem 2.3 if an initial point $v_0 \in \bigcap_{C \in \mathcal{C}} C$ is given. The new algorithm is also simpler, as it avoids Lemma 2.2. First, by translation of coordinates, we may assume that v_0 is the origin. For simplicity, suppose that the constraints are non-degenerate and the origin is in the interior of $\bigcap_{C \in \mathcal{C}} C$. For the objective function, it is more convenient to assume that $a = (0, -1)$ rather than $a = (1, 0)$. We thus want to maximize the y-coordinate in the intersection $\bigcap_{C \in \mathcal{C}} C$.

As in Section 2.1, we rephrase the problem in terms of univariate functions. For each convex set $C \in \mathcal{C}$, we construct a convex function $h : \mathbb{R} \to \mathbb{R}$ as follows. Observe that the portion of the boundary of C below the x-axis does not affect the solution to the convex program. Let γ denote the portion of the boundary of C strictly above the x-axis then. The transformation $(x, y) \mapsto (x/y, 1/y)$ maps γ into the graph of a convex function. We define h to be this function. Now, h satisfies the following property: given a point (x, y) with $y > 0$, (x, y) belongs to C if and only if $(x/y, 1/y)$ is above h. Our problem is equivalent to the following:

> Given a collection H of k convex functions on $\mathbb{R}$, find $x \in \mathbb{R}$ minimizing $\max_{h \in H} h(x)$— that is, find the lowest point on the upper envelope of H.

Remark: If the functions in H are linear, then the problem can of course be solved in linear time. Reichling [24] described a linear-time algorithm for the quadratic case. The dual of the problem is equivalent to finding a *bridge* [16] of a convex hull of k planar convex objects. Nielsen and Yvinec [22] gave an $O(k)$-time algorithm for the case when two objects may intersect at most a constant number of times. We know of no nontrivial deterministic algorithm for the general case. Chazelle and Matoušek [7] specifically pointed out that their derandomization technique is not applicable to this problem in general, because the range space associated with H does not necessarily have bounded *Vapnik-Chervonenkis (VC) dimension*.

We now solve the problem using the following primitive operations:

(F1) Given $h_1, h_2 \in H$, find $x \in \mathbb{R}$ minimizing $\max\{h_1(x), h_2(x)\}$.

(F2) Given $h \in H$ and $x \in \mathbb{R}$, evaluate $h(x)$.

For the functions we have constructed, these operations reduce to operations (D1) and (D2); in fact, operation (D1) can be replaced by the weaker operation (R1).

It turns out that the convexity of the functions H is not crucial: it suffices to assume that each function $h \in H$ is upward unimodal. Compute $x_{\min}$ that minimizes h using operation (F1). We can define a decreasing function $f : \mathbb{R} \to \mathbb{R}$ such that within $(-\infty, x_{\min}]$, f coincides with h, and within $(x_{\min}, \infty)$, f is linear with slope approaching $-\infty$. Similarly, we can define an increasing function $g : \mathbb{R} \to \mathbb{R}$ such that within $[x_{\min}, \infty)$, g coincides with h, and within $(-\infty, x_{\min})$, g is linear with slope approaching ∞. Then our problem becomes:

> Given a collection F of k decreasing functions and a collection G of k increasing functions, find x minimizing $\max_{\phi \in F \cup G} \phi(x)$.

We denote the optimal value of x by x^*.

In general, two increasing functions or two decreasing functions may intersect an arbitrary number of times, but an increasing and a decreasing function intersect at most once. The following operations reduce to (F1) and (F2):

(F1$'$) Given $f \in F$ and $g \in G$, find the unique $x \in \mathbb{R}$ with $f(x) = g(x)$.

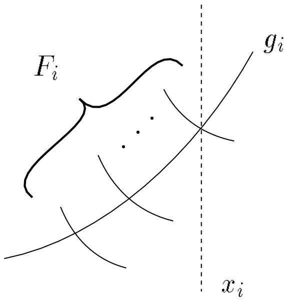

Figure 3: Prune F_i if $x^* > x_i$. Prune g_i if $x^* < x_i$.

(F2′) Given $\phi \in F \cup G$ and $x \in \mathbb{R}$, evaluate $\phi(x)$.

Theorem 2.6 *A convex program in* $\mathbb{R}^2$ *with* k *constraints and a linear objective function can be solved using* $O(k \log k / \log\log k)$ *primitive operations if an initial feasible point is given.*

Proof: As before, let $p = |F|$ and $q = |G|$; initially, $p = q = k$. Without loss of generality, assume $p \geq q$. Partition $F = F_0 \cup \cdots \cup F_q$ such that $|F_1| = \cdots = |F_q| = \lfloor p/q \rfloor$. Let $G = \{g_1, \ldots, g_q\}$. For each $i = 1, \ldots, q$, compute the x-coordinate of the intersection of g_i with each function in F_i and let x_i denote the largest of these $\lfloor p/q \rfloor$ x-coordinates. Observe the following property concerning x_i: if $x^* > x_i$, then $\max_{f \in F_i} f(x^*) < g_i(x^*)$; conversely, if $x^* < x_i$, then $\max_{f \in F_i} f(x^*) > g_i(x^*)$. In the former case, F_i can be pruned; in the latter, g_i can be pruned. See Figure 3.

Using a linear-time selection algorithm, compute the $\lceil \alpha q \rceil$ smallest number in $\langle x_1, \ldots, x_q \rangle$ and denote it by x. Here, $\alpha \in (0,1)$ is a fixed parameter to be determined later (as we will see, $\alpha = 1/2$ is not the best choice). In linear time, we can compare the value of x^* with x by comparing $\max_{f \in F} f(x)$ with $\max_{g \in G} g(x)$. If $x^* = x$, then we are done. If $x^* > x$, then we can prune F_i from F for each $x_i \leq x$, and as a result, the size of F reduces to at most $p - \alpha q \lfloor p/q \rfloor < p - \alpha q(p/(2q)) = (1 - \alpha/2)p$. If $x^* < x$, then we can prune g_i from G for each $x_i \geq x$, and as a result, the size of G reduces to at most $q - (1-\alpha)q = \alpha q$.

By repeating the process for the new sets of functions F and G, we arrive at the following recurrence for the running time $T(p,q)$ for p decreasing functions and q increasing functions, where A is the time to perform a primitive operation:

$$T(p,q) \leq \begin{cases} \max\{T(\lfloor (1-\frac{\alpha}{2})p \rfloor, q), T(p, \lfloor \alpha q \rfloor)\} + O(Ap) & \text{if } p \geq q \\ T(q,p) & \text{if } q > p \end{cases}$$

Note that $T(p,0) = T(0,q) = O(1)$, as the solution is unbounded if either F or G is empty. By induction, one can then show that

$$T(p,q) = O(A(p \log_{1/\alpha} q + q \log_{1/\alpha} p + p/\alpha + q/\alpha)).$$

Setting $\alpha = 1/\log^{\varepsilon} k$ for some constant $0 < \varepsilon < 1$ implies that $T(k,k) = O(Ak \log k / \log\log k)$. $\square$

3 An Algorithm for Convex n-gons

In this section, we discuss an instance of our convex programming problem where the constraints are convex n-gons. As both operations (D1) and (D2) can be carried out in $O(\log n)$ time [6, 10, 25],

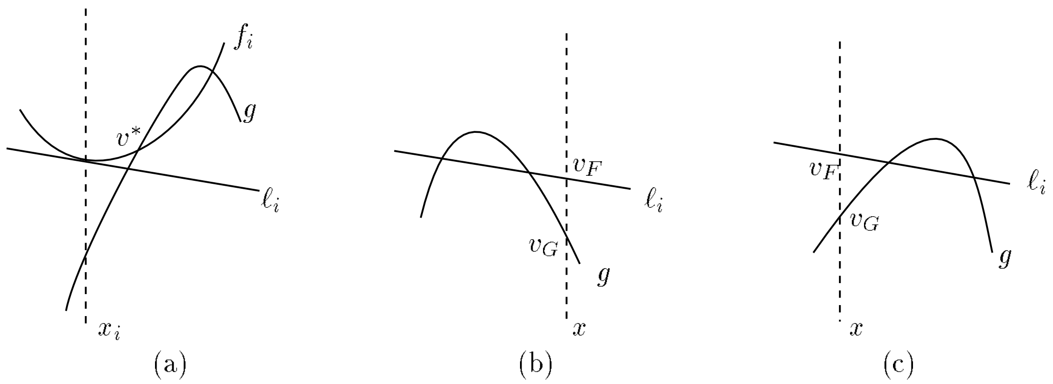

Figure 4: (a) In Case 1, a contradiction is reached if the right half of f_i defines v^*. (b,c) In Case 2, simple tests determine whether v^* is to the left or to the right of x.

Theorem 2.3 yields an $O(k \log k \log n)$-time algorithm. Using additional ideas, we give an algorithm with an $O(k \log n \log\log n)$ running time, which is more efficient for $k \gg \log n$.

We follow the framework from Section 2.1 and consider the problem: given a collection F of p convex functions and a collection G of q concave functions on $\mathbb{R}$, find the left intersection point v^* of $\max_{f \in F} f$ and $\min_{g \in G} g$. The functions in $F \cup G$ are now piecewise-linear, each consisting of at most n links. In what follows, we assume that the sequence of links of each function is stored in an array. It is straightforward to modify the algorithm if the links of each function are stored instead in a tree structure of logarithmic depth.

We first give an efficient algorithm for the case $p \gg q$, inspired by the prune-and-search method of Reichling [24]. The algorithms in Section 2 prune entire functions; here, we prune portions of functions. Reichling's method yields a running time of $O(p \log^2 n + q \log^2 n)$. We use new geometric insights to remove a logarithmic factor in the first term.

Lemma 3.1 *The above problem can be solved in $O(p \log n + q \log^2 n)$ time.*

Proof: Let N be the total number of links in F; initially $N = pn$. Let $F = \{f_1, \ldots, f_p\}$. For each function f_i, let v_i be a point on the graph of f_i that divides f_i into two portions each with roughly half the number of links. Let x_i be the x-coordinate of v_i. The *left (or right) half* of f_i refers to the portion of f_i strictly to the left (or right) of x_i. Let ℓ_i be a tangent of f_i passing through v_i.

Using a linear-time algorithm for weighted medians, one can find an $x \in \mathbb{R}$ such that both the total number of links in $\{f_i : x_i \le x\}$ and the total number of links in $\{f_i : x_i \ge x\}$ are at least $N/2$. Compute the intersection v_F of the upper envelope of $\{\ell_1, \ldots, \ell_p\}$ with the vertical line at x in $O(p)$ time. Compute the intersection v_G of $\min_{g \in G} g$ with the vertical line at x in $O(q \log n)$ time.

Case 1. v_F is below v_G. For each $x_i \ge x$, we claim that the right half of f_i does not define v^* and can thus be pruned. To see this, assume the contrary and suppose v^* is defined by the right half of the convex function f_i as well as the concave function $g \in G$. It is easy to see that the tangent ℓ_i must lie strictly above g at any vertical line to the left of x_i (see Figure 4(a)). This implies that v_F is strictly above v_G, a contradiction.

Case 2. v_F is above v_G. Suppose v_F and v_G are defined by the tangent ℓ_i and the concave function $g \in G$. Then v^* is contained in the region bounded by ℓ_i and g, which is either completely to the left or completely to the right of x (see Figure 4(b,c)). In the former case, the right half of f_i for each $x_i \geq x$ can be pruned; in the latter case, the left half of f_i for each $x_i \leq x$ can be pruned.

In one iteration, we thus remove roughly half the number of links in $\{f_i : x_i \leq x\}$ or $\{f_i : x_i \geq x\}$. The total number of remaining links is $3N/4 + O(p)$. Repeating the process $O(\log n)$ times reduces this number to $O(p)$. Therefore, in $O((p + q \log n) \log n)$ time, we can replace F with a set of $O(p)$ linear functions.

To finish off, we interchange the roles of F and G and reduce G to a set of $O(q)$ linear functions by a similar algorithm. This now requires $O((p+q) \log n)$ time, because intersecting $\max_{f \in F} f$ with a vertical line takes only $O(p)$ time for the set of linear functions F. Finally, the solution v^* can be computed by linear programming in $O(p+q)$ time. $\square$

Theorem 3.2 *A convex program in $\mathbb{R}^2$ with k constraints and a linear objective function can be solved in $O(k \log n \log\log n)$ time if each constraint is a convex n-gon whose sequence of vertices is stored in an array.*

Proof: Fix a parameter R. We follow the algorithm of Theorem 2.3, except when $q \leq R$, we switch to the algorithm of Lemma 3.1. The running time satisfies the following recurrence, as a primitive operation takes $A = O(\log n)$ time:

$$T(p,q) \leq \begin{cases} \max\{T(\left\lfloor \frac{7}{8}p \right\rfloor, q), T(p, \left\lfloor \frac{15}{16}q \right\rfloor)\} + O(p \log n) & \text{if } p \geq q > R \\ O(p \log n + q \log^2 n) & \text{if } p \geq q \text{ and } q \leq R \\ T(q,p) & \text{if } q > p \end{cases}$$

We thus have $T(k,k) = O(k \log n \log(k/R) + R \log^2 n)$. Setting $R = k/\log n$ yields $T(k,k) = O(k \log n \log\log n)$. $\square$

The observations in Section 2.3 imply:

Corollary 3.3 *We can decide whether k convex n-gons in the plane have a common intersection in $O(k \log n \log\log n)$ time.*

4 Online Linear Programming in $\mathbb{R}^3$

In this section, we apply the techniques from Section 2 to the following problem: store a set H of $\leq n$ halfspaces in $\mathbb{R}^3$ into a data structure so that we can quickly find the point in $\mathcal{P} = \bigcap_{h \in H} h$ minimizing a given linear objective function. With Dobkin and Kirkpatrick's *hierarchical representations* of convex polyhedra [11, 12], the problem can be solved in $O(\log n)$ time using a linear-size data structure. Here, we are interested in a data structure that can support insertions of new halfspaces. We solve this problem in $o(\log^3 n)$ worst-case time per query and insertion.

4.1 The fixed-objective case

We first consider the case when the same linear objective function is used in all the queries. Without loss of generality, we assume that the objective is to minimize the z-coordinate.

We borrow a general technique by Bentley and Saxe [2] for transforming a static data structure into a data structure that supports insertion. This technique is designed for *decomposable search problems*, but as noted by Eppstein [15], it is also applicable to linear programming if one can efficiently optimize linear functions over an intersection of preprocessed polyhedra.

Theorem 4.1 *The minimum z-coordinate in an intersection of $\leq n$ halfspaces in $\mathbb{R}^3$ can be maintained in $O(\log^3 n/\log\log n)$ worst-case time under a sequence of insertions.*

Proof: Fix an integer $b > 1$. Let $a_{k-1}\ldots a_1 a_0$ be a base-b representation of $|H|$, where $k \leq \log_b n+1$. Partition H into k subsets $H_0, \ldots, H_{k-1}$ such that $|H_i| = a_i b^i$. Our data structure consists of hierarchical representations of the k polyhedra $\mathcal{P}_i = \bigcap_{h\in H_i} h$ $(i = 0, \ldots, k-1)$.

To insert a new halfspace h into our data structure, let j be the smallest index with $a_j \neq b-1$. In the new base-b representation of $|H|$, the number a_j is incremented and $a_0, \ldots, a_{j-1}$ are set to zero. Consequently, we set $H_j \leftarrow H_0 \cup \cdots \cup H_j \cup \{h\}$ and set $H_0, \ldots, H_{j-1}$ to $\emptyset$. The hierarchical representations of the corresponding polyhedra need to be updated; this can be done in $O(b^{j+1})$ time using Chazelle's polyhedra intersection algorithm [5]. One can show that the amortized insertion time is $O(b \log n/\log b)$.

Let $v_{\min}$ be the point that minimizes the z-coordinate in $\mathcal{P}$ before h is inserted. We now describe how to compute the new optimum $v'_{\min}$ after the insertion using the above data structure. First if $v_{\min} \in h$, then $v'_{\min} = v_{\min}$. Otherwise, $v'_{\min}$ must lie on the bounding plane π of h. We can compute $v'_{\min}$ by minimizing a linear function over $\bigcap_{i=0}^{k-1}(\mathcal{P}_i \cap \pi)$. Since $\{\mathcal{P}_i \cap \pi\}$ is a collection of k convex sets in a two-dimensional plane, Theorem 2.4 can be used to solve the convex program.

Primitive operation (D1) requires minimizing a linear function over the intersection of two polyhedra and a plane; using hierarchical representations, it can be done in $A_1 = O(\log^2 n)$ time (e.g., see the paper by Eppstein [15]). Primitive operation (D2) requires intersecting a polyhedron with a line; it can be done in $A_2 = O(\log n)$ time. Thus, the time needed to update the optimum is $O(A_1 k + A_2 k \log k) = O(\log^3 n/\log b)$.

Setting the parameter $b = \lfloor \log n \rfloor$, we obtain an $O(\log^3 n/\log\log n)$ amortized time bound for maintaining $v_{\min}$ under a sequence of insertions. This amortized bound can be made worst-case using known techniques for decomposable search problems [23]. $\square$

Note that a method for performing operation (D1) in $o(\log^2 n)$ time will directly improve the time bound in the above theorem.

As an application of Theorem 4.1, we have an online algorithm for the largest circle (in fact, the largest homothet of any "simply-shaped" convex figure) inside an intersection of halfplanes in $\mathbb{R}^2$. Previously, Boissonnat et al. [3] gave static data structures for finding the largest circle in a convex polygon subject to point/line constraints.

Corollary 4.2 *The largest circle contained in an intersection of $\leq n$ halfplanes in $\mathbb{R}^2$ can be maintained in $O(\log^3 n/\log\log n)$ worst-case time under a sequence of insertions.*

Proof: A circle with center (x, y) and radius z is contained in a halfplane $\{(\xi, \eta) : a\xi + b\eta \leq 1\}$ if and only if $ax + by + (\sqrt{a^2+b^2})\, z \leq 1$. Hence, finding the largest circle contained in the intersection of halfplanes in $\mathbb{R}^2$ reduces to finding the largest z-coordinate in an intersection of halfspaces in $\mathbb{R}^3$. $\square$

4.2 The general case

We now consider the general case in which each query may use a different linear objective function. Instead of the hierarchical representation, we find it more convenient in this instance to use the *drum representation* of a polyhedron [6, 10]. Let $\mathcal{Q}$ be a convex polyhedron with n vertices, and for simplicity, assume that no two vertices have the same z-coordinate. Let $z_1 < \cdots < z_n$ denote the z-coordinates of the vertices of $\mathcal{Q}$. The i-th drum consists of the ordered sequence of faces of $\mathcal{Q}$ that intersect the horizontal plane $\{(x, y, z) : z = (z_i + z_{i+1})/2\}$ $(i = 0, \ldots, n-1)$. The drum representation is the collection of these $n-1$ drums.

In the drum representations, the ordered sequence for each drum need not be stored explicitly; otherwise, $\Omega(n^2)$ storage is necessary. Instead, we only assume that we can perform binary searches on the sequence in logarithmic time. Using persistent search trees [26], it is possible to construct a drum representation with $O(n)$ storage in $O(n \log n)$ time.

Theorem 4.3 *The online linear programming problem can be solved in $O(\log^3 n/\log\log\log n)$ time per query and $O(\log^3 n/\log\log n)$ time per insertion.*

Proof: Let $v_{\min}$ and $v_{\max}$ be the points that minimize and maximize the z-coordinate in $\mathcal{P}$ respectively; let $z_{\min}$ and $z_{\max}$ denote their z-coordinates. By Theorem 4.1, they can be maintained in $O(\log^3 n/\log\log n)$ time.

We use the same data structure as in Theorem 4.1, except that besides the hierarchical representations of the $\mathcal{P}_i$'s, we also store the drum representations of the $\mathcal{P}_i$'s. The time needed to insert a new halfspace into the data structure is now $O(b \log^2 n/\log b)$ instead of $O(b \log n/\log b)$, since the preprocessing time for the drum representation is larger by a logarithmic factor.

To answer a query, we need to optimize a given linear function over $\mathcal{P} = \bigcap_{i=0}^{k} \mathcal{P}_i$. Let z^* be the z-coordinate of the optimal point. We first describe how to solve the decision problem: given a number $c \in \mathbb{R}$, decide whether $z^* < c$, $z^* = c$, or $z^* > c$. We may assume that $c \in (z_{\min}, z_{\max})$; otherwise, the solution is trivial.

It is not difficult to see that the decision problem reduces to the problem of minimizing a linear function over $\bigcap_{i=0}^{k}(\mathcal{P}_i \cap \pi)$, where π denotes the horizontal plane $\{(x, y, z) : z = c\}$. This is a two-dimensional convex programming problem. Furthermore, we know a feasible point for the convex program, namely, the intersection of the line segment $\overline{v_{\min} v_{\max}}$ with the plane π. Thus, Theorem 2.6 yields an algorithm with $O(k \log k/\log\log k)$ primitive operations.

The drum representations allow us to identify the ordered sequence of edges of the polygon $\mathcal{P}_i \cap \pi$ in logarithmic time. Thus, operations (D1) and (D2) can both be carried out in $O(\log n)$ time. We conclude that the decision problem can be solved in $O(k \log k \log n/\log\log k)$ time.

To find z^*, we apply Megiddo's parametric search technique [18]. Suppose that there is a sequential algorithm solving the decision problem in T_s time as well as a parallel algorithm solving the decision problem in T_p time with P processors. Then for a general class of parallel algorithms, the parametric search technique finds z^* in $O(PT_p + T_s T_p \log P)$ time. Cole [9] showed that for a restricted class of parallel algorithms, this running time can be improved to $O(PT_p + T_s(T_p + \log P))$.

Instead of parallelizing the algorithms in Section 2, we use a trivial parallel algorithm for our decision problem. In the framework of Section 2.6, this algorithm does the following: (i) for each $f \in F$ and $g \in G$, it computes the intersection point of f and g; (ii) then it finds the optimum, which is the intersection point with the largest y-coordinate. Step (i) can be done in $O(\log n)$ time using

k^2 processors, since a primitive operation takes logarithmic time. Step (ii) can be done in $O(\log k)$ steps using the standard "binary-tree" algorithm for finding the maximum. Megiddo's parametric search technique now yields an algorithm with running time

$$O\left(k^2 \log(nk) \;+\; \left(\frac{k \log k \, \log n}{\log\log k}\right) \log(nk) \log k\right).$$

Cole's improvement can be applied to reduce a $\log k$ factor from the second term. Substituting $b = \lfloor \log n \rfloor$ and $k = O(\log_b n)$, we see that z^* can be computed in $O(\log^3 n / \log\log\log n)$ time. □

Remarks:

1. With drum representations and parametric search, we can optimize a linear function over the intersection of three convex polyhedra in $O(\log^2 n)$ time (as the decision problem can be solved in $O(\log n)$ time). Previously, Eppstein [15] obtained an $O(\log^3 n)$-time bound using hierarchical representations. If we use this result in combination with Clarkson's randomized algorithm [8] for convex programming as was suggested by Eppstein, we can solve the three-dimensional online linear programming problem with expected $O(\log^2 n \log\log n)$ time per query and $O(\log^2 n)$ time per insertion.

2. Reichling [25] applied similar ideas (drum representations, parametric search, etc.) to detect whether k preprocessed convex polyhedra have a common intersection. His method requires $O(k \log k \log^3 n)$ time. We can solve this problem in $O(k \operatorname{polylog} k \log^2 n)$ time by following the algorithm of Theorem 4.3, but using a more efficient parallelization of the algorithm of Theorem 2.6 (with $O(k)$ processors and $O(\operatorname{polylog} k \log n)$ steps). Alternatively, with the earlier remark, Clarkson's randomized algorithm solves this problem in $O(k \log n + \log k \log^2 n)$ expected time.

5 Conclusions

In this paper, we have given general deterministic algorithms for the two-dimensional convex programming problem. We have applied our techniques to obtain improved worst-case time bounds for detecting a common intersection in a collection of convex polygons, as well as solving online versions of the linear programming problem in three dimensions.

Many open problems still remain. Perhaps the most interesting is to determine whether linear worst-case complexity is attainable for two-dimensional convex programming, using only the primitive operations provided in Section 2. Another problem is to formulate general classes of convex programs in higher dimensions that admit near-linear deterministic algorithms. Finally, to what extent can we improve the results in Sections 3 and 4?

Acknowledgement

I thank Jack Snoeyink for suggesting the problem in Section 3.

References

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

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

[3] J.-D. Boissonnat, J. Czyzowicz, O. Devillers, and M. Yvinec. Circular separability of polygons. In *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms*, pages 273–281, 1995.

[4] T. M. Chan. Fixed-dimensional linear programming queries made easy. In *Proc. 12th ACM Sympos. Comput. Geom.*, pages 284–290, 1996.

[5] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21:671–696, 1992.

[6] B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *J. ACM*, 34:1–27, 1987.

[7] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21:579–597, 1996.

[8] K. L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42:488–499, 1995.

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

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

[11] D. P. Dobkin and D. G. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *J. Algorithms*, 6:381–392, 1985.

[12] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra: A unified approach. In *Proc. 17th Int. Colloq. Automata, Languages, and Programming*, Lect. Notes in Comput. Sci., vol. 443, Springer-Verlag, pages 400–413, 1990.

[13] M. E. Dyer. Linear time algorithms for two- and three-variable linear programs. *SIAM J. Comput.*, 13:31–45, 1984.

[14] M. E. Dyer. A class of convex programs with applications to computational geometry. In *Proc. 8th ACM Sympos. Comput. Geom.*, pages 9–15, 1992.

[15] D. Eppstein. Dynamic three-dimensional linear programming. *ORSA J. Comput.*, 4:360–368, 1992.

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

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

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

[19] N. Megiddo. Linear time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput.*, 12:759–776, 1983.

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

[21] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice-Hall, Englewood Cliffs, N.J., 1993.

[22] F. Nielsen and M. Yvinec. An output-sensitive convex hull algorithm for planar objects. Research Report 2575, INRIA, Sophia-Antipolis, France, 1995.

[23] M. H. Overmars and J. van Leeuwen. Dynamization of decomposable searching problems yielding good worst-case bounds. In *Proc. 5th GI Conf. Theoret. Comput. Sci.*, Lect. Notes in Comput. Sci., vol. 104, Springer-Verlag, pages 224–233, 1981.

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

[25] 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, Springer-Verlag, pages 180–186, 1988.

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

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

[28] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proc. 9th Sympos. Theoret. Aspects Comput. Sci.*, Lect. Notes in Comput. Sci., vol. 577, Springer-Verlag, pages 569–579, 1992.

[29] S. Toledo. Maximizing non-linear concave functions in fixed dimension. In *Proc. 33rd IEEE Sympos. Found. Comput. Sci.*, pages 696–685, 1992.