SERIE B — INFORMATIK

On linear-time deterministic algorithms for optimization problems in fixed dimension

Bernard Chazelle*
Jiří Matoušek**

B 92–18
September 1992

Abstract

We show that with recently developed derandomization techniques, one can convert Clarkson's randomized algorithm for linear programming in fixed dimension into a linear-time deterministic one. The constant of proportionality is $d^{O(d)}$, which is better than for previously known such algorithms. We show that the algorithm works in a fairly general abstract setting, which allows us to solve various other problems (such as finding the maximum volume ellipsoid inscribed into the intersection of n halfspaces) in linear time.

Keywords: Computational complexity, computational geometry, geometric optimization problem, randomized algorithm, derandomization

*Supported in part by NSF Grant CCR-90-02352 and the Geometry Center. Department of Computer Science Princeton University Princeton, NJ 08544, USA.
**Supported in part by Humboldt Research Fellowship. Department of Applied Mathematics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czechoslovakia, and Free University of Berlin, Arnimallee 2–6, W-1000 Berlin 33, Germany.

1 Introduction

We consider the *linear programming problem*: given a set H of n halfspaces in R^d and a direction vector $c \in R^d$, find a point $x \in \bigcap H$ that minimizes $c.x$. We restrict ourselves to the case where n is very large compared to d, and so, the dependency of the running time on n is of primary interest. For the algorithms we will discuss the running time will be linear in n for any *fixed* d, with a constant of proportionality exponential in d: It is this dependency on d which we wish to study closely.

There are known polynomial-time algorithms for linear programming [Kha80, Kar84], but the number of arithmetic operations they perform depends on the bit complexity of the input and grows with the precision of the numbers describing the input. In contrast, we consider purely combinatorial algorithms, where the number of arithmetic operations performed only depends on n, d and not on the precision of the input numbers. A natural model of computation for these considerations is the *real RAM*, where the input data contain arbitrary real numbers and each arithmetic operation with real numbers is charged unit cost[1].

Megiddo [Meg84] has given the first deterministic algorithm whose running time is of the form $O(C(d)n)$, with $C(d) = 2^{2^d}$. This was improved to $C(d) = 3^{d^2}$ by Dyer [Dye86] and Clarkson [Cla86]. Recently, a number of randomized algorithms have been presented for the problem, see [DF87], [Cla88], [Sei91], with a better dependency on d. Clarkson's algorithm [Cla88] has the best expected complexity among those, namely $O(d^2 n + d^{d/2+O(1)} \log n)$. Notice that although this is still exponential in d the exponential term is multiplied only by a $\log n$ factor. Recently Kalai [Kal92] and independently Matoušek, Sharir and Welzl [MSW92] have developed algorithms with a subexponential dependency on both n and d. In combination with Clarkson's algorithm, one obtains a randomized algorithm for linear programming with expected running time $O(d^2 n + e^{O(\sqrt{d \ln d})} \log n)$. To match these performance bounds by a deterministic algorithm seems to be difficult at the present time.

However, as was observed by the authors of this paper some time ago and mentioned in [Cha91], one can apply the derandomization technique of [Mat91] to the above mentioned Clarkson's randomized algorithm and obtain another linear-time deterministic algorithm for linear programming. We prove here that the constant of proportionality is of the form $d^{O(d)}$, which is far behind the randomized complexity, but significantly better than the constants for the previously known deterministic algorithms.

Clarkson's algorithm can be shown to work in a general framework, which includes various other geometric optimization problems (see below for examples). With few extra algorithmic assumptions, our derandomization works in this framework as well. For some of these problems, linear-time deterministic algorithms were given by Dyer [Dye92]; our approach again brings a better dependency on the dimension. For others, as e.g., finding the maximum volume ellipsoid inscribed into a polyhedron in R^d (given by its n facets), we get the first known efficient determinis-

[1] As for the bit complexity, one can show that the bit size of the numbers is at most a constant multiple of the bit size of the input numbers in the algorithms we will consider.

tic algorithm. For some of these problems there are much more efficient randomized algorithms known – Clarkson's algorithm itself (applied in this setting), and a recent subexponential algorithm by Gärtner [Gär92].

In this paper we first describe the tools used for derandomization. We use this opportunity to give a somewhat different and hopefully simpler presentation of an algorithm from [Mat91], and we will be careful to trace the dependence of the constants of d. We will also outline a parallel (NC) version of the algorithm. Then we give a deterministic variant of Clarkson's algorithm, and discuss sample problems where it can be applied.

Throughout the paper, the $O()$ symbol only hides absolute constants, independent of the dimension d. If a dependency of the constant on d is allowed, then we write $O_d()$.

2 Computing ε-approximations and ε-nets

We begin by briefly recalling some definitions; we refer e.g., to [HW87],[Mat91] for more details. Let $\Sigma = (X, \mathcal{R})$ be a set system[2] on a set X. If Y is a subset of X, we denote by $\mathcal{R}|_Y$ the set system $\{R \cap Y;\ R \in \mathcal{R}\}$ (the system *induced by* $\mathcal{R}$ *on* Y; let us emphasize that although many sets of $\mathcal{R}$ may intersect Y in the same subset, this intersection only appears once in $\mathcal{R}|_Y$).

Let us say that a subset $Y \subseteq X$ is *shattered* (by $\mathcal{R}$) if every possible subset of Y is induced by $\mathcal{R}$, i.e. if $\mathcal{R}|_Y = 2^Y$. We define the *Vapnik-Chervonenkis dimension*, VC-*dimension* for short, of the set system $\Sigma = (X, \mathcal{R})$ as the maximum size of a shattered subset of X (if there are shattered subsets of any size, then we say that the VC-dimension is infinite). Let us define the *shatter function* $\pi_{\mathcal{R}}$ of $(X, \mathcal{R})$ as follows: $\pi_{\mathcal{R}}(m)$ is the maximum possible number of sets in a subsystem of $(X, \mathcal{R})$ induced by an m point subset of X. The shatter function of a set system of VC-dimension d is bounded by $\binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{d}$ ([Sau72], [VC71]), and conversely, if the shatter function is bounded by a fixed polynomial, then the VC-dimension is bounded by a constant.

Set systems of VC-dimension bounded by a constant occur naturally in geometry; a typical example is the plane standing for X and the set of all triangles for $\mathcal{R}$.

The following two notions will be crucial in our algorithm. Let X be finite, $\varepsilon \in (0, 1)$. A subset $A \subseteq X$ is a *ε-approximation* for $(X, \mathcal{R})$, provided that

$$\left| \frac{|A \cap R|}{|A|} - \frac{|R|}{|X|} \right| \leq \varepsilon$$

for every set $R \in \mathcal{R}$. A subset $S \subseteq X$ is called an *ε-net for* $(X, \mathcal{R})$ provided that $S \cap R \neq \emptyset$ for every set $R \in \mathcal{R}$ with $|R|/|X| > \varepsilon$. In the sequel, it will be more convenient to write $1/r$ for ε, with $r > 1$.

[2] In computational geometry literature, set systems in this context are usually called *range spaces*, the sets belonging to the set system *ranges* and the elements of the underlying set *points*. In this paper we will not use this terminology.

A remarkable property of set systems of VC-dimension at most d is that for any $r > 1$, they admit a $(1/r)$-approximation whose size only depends on d and r, *not* on the size of X; the quantitative bounds on the size are $(1+o(1))dr \log r$ for $(1/r)$-nets [KPW92], and $O_d(r^{2-2/(d+1)}(\log r)^{2-1/(d+1)})$ for $(1/r)$ approximations [MWW92].

These $(1/r)$ nets and $(1/r)$-approximations of small size are quite powerful tools for derandomizing geometric algorithms. It was shown in [Mat91] that under certain computational assumptions about the set system $(X, \mathcal{R})$ of a bounded VC-dimension, one can compute small $(1/r)$-approximations and $(1/r)$-nets for small r reasonably efficiently, in time $O_d((r^{2d} \log^d r)|X|)$ (in particular, in linear time for fixed d, r). In this section we give a somewhat simplified exposition of this result, using observations of [MWW92], and we will estimate the dependence on d more carefully.

The algorithm will work by a repeated application of the following "halving lemma". It is based on standard result from discrepancy theory:

Lemma 2.1 *Let $(A, \mathcal{S})$ be a set system, $n = |A|$, $m = |\mathcal{S}|$, n even. Then one can find, in $O(nm)$ deterministic time, a subset $\bar{A} \subset A$ of size $|A|/2$, which is an ε-approximation for $(A, \mathcal{R})$ with*

$$\varepsilon \leq \sqrt{\frac{8 \ln(4m+4)}{n}}.$$

Proof: We let $\mathcal{S}' = \mathcal{S} \cup \{A\}$, and we find, following the method nicely described in Spencer's book [Spe87], a mapping $\chi : A \to \{-1, +1\}$, such that $|\chi(S)| \leq \Delta = \sqrt{2n \ln(4|\mathcal{S}'|)}$ for every $S \in \mathcal{S}'$. One uses a standard probabilistic proof showing that a random mapping χ works with probability at least $1/2$, and then derandomizes it using the method of conditional probabilities. With some care, the derandomized algorithm can be implemented to run in $O(nm)$ time, see [Mat91].

With such a χ, we let A' be the larger of the sets $\chi^{-1}(1)$, $\chi^{-1}(-1)$. Since $A \in \mathcal{S}'$, we have $|A'| - |A \setminus A'| \leq \Delta$, or $|A'| - \frac{n}{2} \leq \frac{\Delta}{2}$. We remove $|A'| - \frac{n}{2}$ arbitrary elements of A', forming a set $\bar{A}$ with exactly $n/2$ elements. Then for a $S \in \mathcal{S}$, we have $|\,2|\bar{A} \cap S| - |S|\,| \leq |\,2|A' \cap S| - |S|\,| + 2(|A'| - |\bar{A}|) \leq 2\Delta$. Thus

$$\left| \frac{|\bar{A} \cap S|}{|\bar{A}|} - \frac{|S|}{|A|} \right| = \frac{1}{n} |\,2|\bar{A} \cap S| - |S|\,| \leq \frac{2\Delta}{n},$$

from which the bound on ε follows. □

For efficient computation of ε-approximations and ε-nets, we need that the set system is given to us in a more "compact" form than by the list of its sets.

Definition 2.2 *We say that a set system $(X, \mathcal{R})$ has a* subsystem oracle of dimension *d, if $\pi_{\mathcal{R}}(m) = O(m)^d$ and there is an algorithm (oracle) which, given a subset $A \subseteq X$, returns the list of sets in $\mathcal{R}|_A$ (each set represented by a list of its members), in $O(|A|)^{d+1}$ time.*

Following the basic scheme of [Mat91], we prove the following:

Theorem 2.3 *Let $(X, \mathcal{R})$ be a set system with a subsystem oracle of dimension d. For a given $r > 1$, one can in time*

$$O(d)^{3d} r^{2d} \log^d(dr)|X|$$

compute a $(1/r)$-approximation of size $O(dr^2 \log(dr))$ for $(X, \mathcal{R})$, and a $(1/r)$-net of size $O(dr \log(dr))$ for $(X, \mathcal{R})$.

Proof: The following technical assumption simplifies the presentation considerably: We suppose that $n = |X|$ is of the form 2^p for an integer p. To remove this assumption is not difficult; we may, for instance, add at most $n/2$ "artificial" points belonging to no set of $\mathcal{R}$ to X, compute a $(1/2r)$-approximation for this new set system, and check that it will also provide a $(1/r)$-approximation for the original system, after removing the artificial points from it; similarly for a $(1/r)$-net.

We describe the algorithm for $(1/r)$-approximations first. It proceeds by partitioning X into subsets of size 2^k each (with k chosen suitably) and then performing a sequence of "halving steps" intermixed with "merging steps". At the beginning of ith step, either a merging or a halving one, we have a current collection $\mathcal{A}_i$ of disjoint subsets of X. Each set of $\mathcal{A}_i$ has the same number of elements n_i (which will be a power of 2). On the beginning of the 1st step, the union of all sets of $\mathcal{A}_1$ is X, and after the last (Kth, say) step, $\mathcal{A}_{K+1}$ will consist of a single set, which will be the desired $(1/r)$-approximation for $(X, \mathcal{R})$.

If the ith step is a merging one, we arbitrarily partition $\mathcal{A}_i$ into pairs ($|\mathcal{A}_i|$ will always be a power of two). For every pair (A_1, A_2) we form the union $A_1 \cup A_2$, and the collection of all these unions will be $\mathcal{A}_{i+1}$. Hence the size of the sets is doubled and their number halved by a merging step.

If the ith step is a halving one, we consider every $A \in \mathcal{A}_i$. We call the subsystem oracle on A, and we obtain the list of all sets in $\mathcal{R}|_A$; we have $m = |\mathcal{R}|_A| = O(n_i)^d$. Then we apply Lemma 2.1 on $(A, \mathcal{R}|_A)$ and we compute a set $\bar{A} \subset A$ with $|A|/2$ elements, which is an $\varepsilon(n_i)$-approximation for $(A, \mathcal{R}|_A)$, where

$$\varepsilon(t) = \sqrt{\frac{8d(\ln t + O(1))}{t}}. \tag{1}$$

Then we set $\mathcal{A}_{i+1} = \{\bar{A};\ A \in \mathcal{A}\}$. Thus a halving step preserves the number of sets and halves their size.

Elementary observations in [Mat91] imply that when the algorithm finishes with a single set, this set will be a ν-approximation for $(X, \mathcal{R})$, where ν is the sum of the $\varepsilon(n_i)$ over all i such that the ith step was a halving step[3].

It remains to show how to organize the sequence of halving and merging steps, so that the running time remains small, and that ν, the resulting error of the approximation, remains below $1/r$. First, we keep alternating halving step and merging

[3]The relevant observations can be summarized in the phrases "an ε-approximation for a δ-approximation is an $(\varepsilon + \delta)$-approximation" and "an ε-approximation for a disjoint union of two sets of equal cardinality can be obtained as the union of ε-approximations of equal cardinality for both sets".

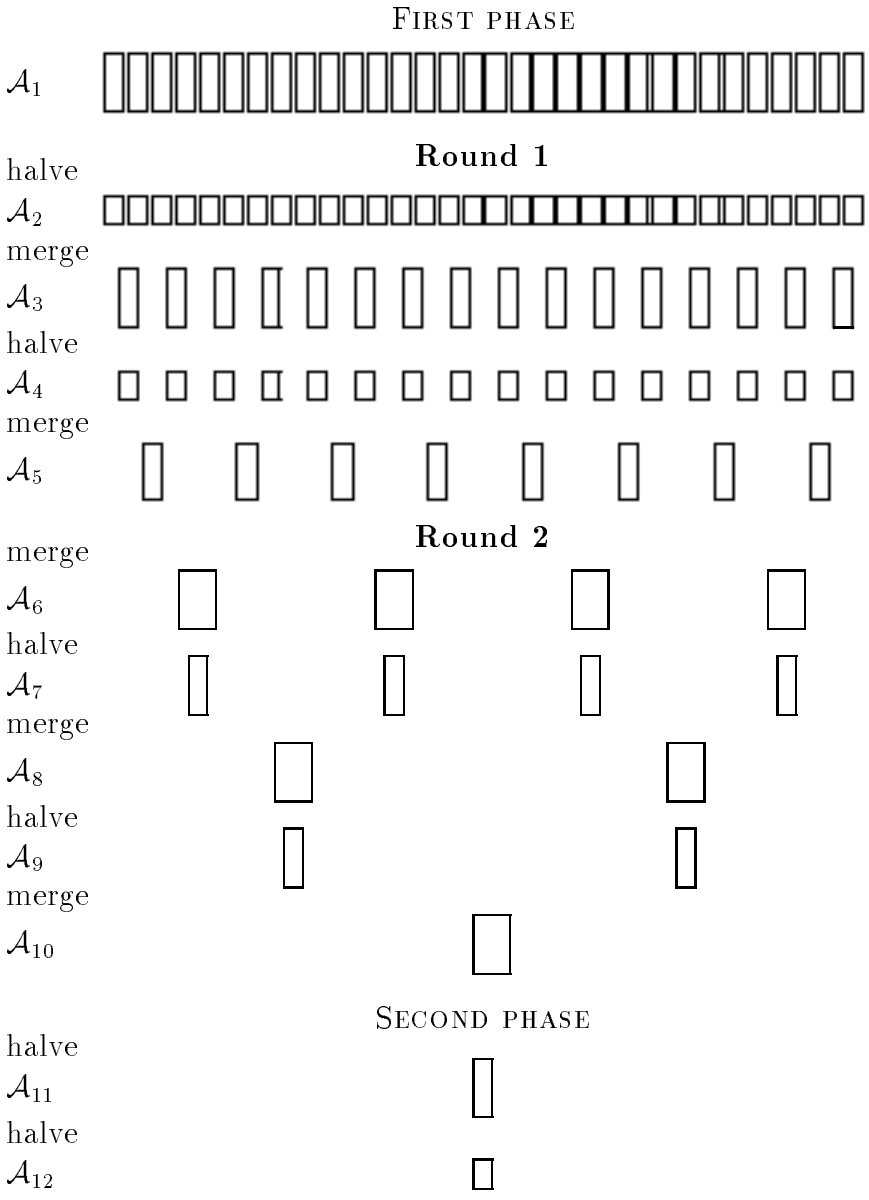


Figure 1: Schematic illustration of the algorithm, with $d = 1$.

step $(d+1)$ times, and after these $2(d+1)$ steps (ending with a merging step) we perform one extra merging step (see Fig. 1). We call this the first round, and the second and further rounds will be exactly the same ($(d+1)$ alternations and one extra merging step). We finish this first phase as soon as there remains only one set in $\mathcal{A}_i$ (thus we cannot perform any more merging steps). Then, in a second phase, we repeat the halving step, until the size of the single current set A becomes so small that the $\varepsilon(|A|)$ given by (1) exceeds $1/8r$. Then we finish, and the resulting set A will be our $(1/r)$-approximation; we must now prove that everything works as claimed.

First consider the size of the resulting set A. By the terminating rule, we must have $\varepsilon(|A|) \geq 1/8r$. From this we calculate $|A| = O(dr^2 \log(dr))$.

We observe that with $t \geq 5$ (say), we have

$$\varepsilon(2t) \leq \frac{3}{4}\,\varepsilon(t)\,. \tag{2}$$

Let us estimate the total error in the approximations, i.e. the $\varepsilon(n_i)$'s summed over all halving steps. The first phase of the algorithms begins with sets of size 2^k (for k to be determined). In the first round one performs $(d+1)$ halving steps with sets of this size, contributing $(d+1)\varepsilon(2^k) = O(d^{3/2}k^{1/2}2^{-k/2})$. Then the size of the current sets is doubled by the extra merging step, and the second round contributes an error of at most $(d+1)\varepsilon(2^{k+1})$, etc. We may assume $n_i \geq 5$ for all i, so by (2) the errors of approximation in the successive rounds form a geometrically decreasing sequence, and the total error in the first phase is thus $O(d^{3/2}k^{1/2}2^{-k/2})$. Choosing a value of k with k with $2^k = Cd^3r^2\log(rd)$ for large enough absolute constant C, we get that the first phase contributes an error at most $1/2r$.

We now consider the second phase. The terminating condition gives that the error at the last step is at most $1/8r$. Each previous step in the second phase halved the size of the current set, and using (2), we get that the errors in the second phase form a geometrically increasing sequence with quotient at least $4/3$. Thus, the total error in the second phase is at most $(1/8r)/(1-3/4) = 1/2r$, so the final set is indeed a $(1/r)$-approximation for $(X, \mathcal{R})$.

Let us analyze the running time. In the first halving step of the algorithm, we perform the computation as in Lemma 2.1 on $|X|/2^k$ sets of size $2^k = O(d^3r^2\log(rd))$ each; the total time needed for this is $O(d)^{3d}r^{2d}\log^d(rd)$. For the first $d+1$ halving steps of the first round, the total number of sets decreases twice by the mergings in between and the size remains the same, so the running times decrease geometrically within one round. The extra merging step doubles the sizes of sets entering the next round, which increases the running time needed for halving for a single set by 2^{d+1}. However, the $(d+2)$ merging steps in the previous round have reduced the total number of sets by 2^{d+2}, so the running time of the following round is at most half of the time for previous round. This shows that the total running time is as claimed in Theorem 2.3.

To compute a $(1/r)$-net rather than a $(1/r)$-approximation, we first compute a $(1/2r)$-approximation A by the previous algorithm and then we compute a $(1/2r)$-

net for $\mathcal{R}|_A$ by the greedy algorithm of Lovász. This is discussed in [CF90], [Mat91] and estimating the dependence on d is a trivial calculation, so we omit further details. □

Remark. For the subsystem oracle of dimension d, it might be somewhat more natural to assume that the shatter function $\pi_{\mathcal{R}}(m)$ is bounded by the maximum possible value for VC-dimension d, that is, $\binom{m}{d} + \binom{m}{d-1} + \cdots + \binom{m}{0} = O(m/d + 1)^d$, rather than just $O(m)^d$. Then, with somewhat more complicated calculations, we could replace the $\log(dr)$ term in the estimates for the $(1/r)$-net and $(1/r)$-approximation size by $\log r$.

Using known techniques, it is relatively straightforward to obtain a parallel version of the algorithm from Theorem 2.3, which might be of interest in some applications. We will only state the result for a *fixed* d:

Theorem 2.4 *Let $(X, \mathcal{R})$ be a set system with a subsystem oracle of dimension d, d fixed, and suppose that the oracle can be implemented in NC. Then one can compute a $(1/r)$-approximation of size $O_{d,\delta}(r^{2+\delta})$ and a $(1/r)$-net of size $O_{d,\delta}(r^{1+\delta})$ for $(X, \mathcal{R})$ in parallel, with at most nr^c processors and $(\log n)^{c'}$ parallel time for any fixed $\delta > 0$ (the constants c, c' depending on d, δ and on the implementation of the oracle). In particular, the computation can be performed in a polylogarithmic parallel time with $O_{d,r}(n)$ processors for every fixed r.*

Proof sketch: By inspecting the proof of Theorem 2.3, we find that there is only one nontrivial part in parallelizing the algorithm, namely the application of Halving lemma (at least if we do not care about the specific power of logarithm in the parallel running time). Here we can use the results of Berger and Rompel [BR91] (or the very similar results of Motwani et al. [MNN89]).

A particular case of their results gives the following: given a set system $(A, \mathcal{S})$ as in Lemma 2.1 and a fixed $\gamma > 0$, one can compute a coloring $\chi : A \to \{-1, +1\}$ with $|\chi(S)| = O_\gamma(n^{1/2+\gamma}\sqrt{\log m})$ for all $S \in \mathcal{S}$ in a polylogarithmic parallel time and with a polynomial number of processors (depending on γ, the dependence being approximately of the form $(m+n)^{1/\gamma}$). This coloring is somewhat worse than the random one, but it is sufficient for our purposes.

We leave the algorithm for computing $(1/r)$-approximations almost without change, only we use the Berger-Rompel algorithm for the halving step, and we adjust the paremeter k (determining the size of the sets enetring the first halving step) suitably. For the error in the halving step, instead of (1) we get $\varepsilon(t) = O_{d,\gamma}(t^{-(1/2-\gamma)}\sqrt{\ln t})$. For any $\gamma < 1/2$, the complexity of the first halving step will dominate the complexity of the whole algorithm. The value of γ then determines both the size of the sets entering this first halving step (and thus the complexity of the whole algorithm) and the size of the resulting $(1/r)$-approximation. Both these sizes will be roughly $(r\sqrt{\log r})^{1/(1/2-\gamma)}$, so the exponent converges to 2 as $\gamma \to 0$.

The $(1/r)$-net is again computed from a $(1/2r)$-approximation. This time we can use e.g., the result of Berger et al. [BRS89] on parallelizing the set covering problem, from which our claim follows. □

Remark. The most important special case of the above theorem is for a fixed r. In such a situation, we can produce an NC algorithm with a linear number of processors without the machinery of [BR91], [MNN89].

If the size of the sets entering each halving step were only polylogarithmic in n, we could simply implement Halving lemma sequentially. Note that the version of the algorithm described in the proof of Theorem 2.3 does not have this property: although the size of the sets is bounded by a constant both in the first and last halving steps, it increases geometrically during the first phase and decreases geometrically during the second phase, reaching a small positive power of n in the middle.

We can modify the first phase, letting the sizes grow more slowly, but in such a way that the errors of the halving steps still form a convergent series. Namely, we may use the following rule: the extra merging step (increasing the size) is inserted at the end of the ith round only if $f(i) > f(i-1)$, where $f(i) = \lfloor 5 \log_2 i \rfloor$. Thus the sizes grow like $2^{\lfloor 5 \log_2 i \rfloor} \propto i^5$. The total error during the first phase is then at most (with $\varepsilon(t)$ given by (1)) $\varepsilon(n_1)+\varepsilon(n_2)+\varepsilon(n_3)+\cdots \le \varepsilon(n_1)+\varepsilon(2^5 n_1)+\varepsilon(3^5 n_1)+\cdots = O(\varepsilon(n_1))(1+1/2^2+1/3^2+\cdots) = O(\varepsilon(n_1))$. Thus the total error during the first phase is still proportional to the error in the first halving step, only the constant becomes larger. In this way, the sets entering the halving step only reach a polylogarithmic size throughout the algorithm, and we get an NC algorithm for any fixed r.

3 Derandomizing Clarkson's algorithm

To simplify our discussion of linear programming, we suppose that

(i) the vector c determining the objective function is vertical, that is, parallel to the x_d-axis, and we look for the lexicographically smallest[4] optimal vertex,

(ii) we only look for nonnegative solutions (to avoid dealing with unbounded solutions and points at infinity),

(iii) the problem has an admissible nonnegative solution.

It is not difficult to relax these assumptions: (i) is without loss of generality (we can rotate the coordinate system so that the optimization direction becomes vertical); for unbounded solutions, we can formally add "constraints at infinity", which will play the role of nonnegativity in (ii), and also nonadmissible problems can be handled easily. See also [Cla88], [Sei91] for a similar discussion.

We begin by introducing an abstract framework for optimization problems similar to linear programming, due to Sharir and Welzl ([SW92], see also [MSW92]). Clarkson's randomized algorithm for linear programming can be formulated and analyzed in this framework, and with one extra axiom, this will also be the case for the derandomized version. Throughout, we will illustrate the abstract concepts on the specific example of linear programming.

[4]Considering x_d the most significant coordinate and x_1 the least significant one.

In the abstract framework, an optimization problem will be a pair (H, w), where H is a finite set, and $w : 2^H \to \mathcal{W}$ is a function with values in a linearly ordered set $(\mathcal{W}, \leq)$. The elements of H will be called the *constraints*, and for a subset $G \subseteq H$, $w(G)$ will be called the *value* of G.

In the linear programming setting, H will be the given set of halfspaces in R^d, and $\mathcal{W}$ will be the set R^d with the lexicographic ordering. For $G \subseteq H$, $w(G)$ will be the optimal nonnegative solution for G (a point in R^d).

In general, we define an *LP-type problem* to be a pair (H, w) as above, satisfying the following two axioms:

Axiom 1. (Monotonicity) For any $F \subseteq G \subseteq H$, $w(F) \leq w(G)$.
Axiom 2. (Locality) For any $F \subseteq G \subseteq H$ with $w(F) = w(G)$ and any $h \in H$, $w(G \cup \{h\}) > w(G)$ implies that also $w(F \cup \{h\}) > w(F)$.

It is easy to check that both axioms hold for linear programming with the above assumptions. Notice that the second axiom follows from uniqueness of the optimal solution (as a point in R^d) and does not require any general position assumptions concerning the constraints.

For an LP-type problem, a *basis* B is a set of constraints with $w(B') < w(B)$ for all proper subsets B' of B. A *basis for* a subset G of H is a basis B with $B \subseteq G$ and $w(B) = w(G)$. So a basis of G is a minimal subset of G with the same value as G. We say that a constraint $h \in H$ *violates* a basis B, if $w(B \cup \{h\}) > w(B)$.

The maximum cardinality of any basis is called the *combinatorial dimension of* (H, w), and it is denoted by $\dim(H, w)$. In the sequel, D will stand for the combinatorial dimension of the considered LP-type problem. Note that the combinatorial dimension is a monotone function: if $F \subseteq G$ then $\dim(F, w) \leq \dim(G, w)$. Linear programming with d variables has combinatorial dimension exactly d (since we exclude infeasible linear programs, where a minimum set of constraints witnessing infeasibility may have $d + 1$ rather than at most d elements).

In order to formulate an algorithm for solving an LP-type problem, we also need some computational assumptions. Clarkson's randomized algorithm requires

Computational assumption 1. (Violation test) Given a basis B and a constraint $h \in H$, decide whether h violates B (and return an error message if the input set B is not a basis).

For linear programming, the violation test as described can be performed in $O(d^3)$ time: use Gauss elimination to find the vertex defined by B; then the violation test with h needs $O(d)$ additional time. One can, however, organize the algorithm in such a way that the vertex is available together with the basis at no extra cost, then a violation test needs $O(d)$ time only.

The reader familiar with Clarkson's algorithm knows that one also needs to solve "small" subproblems directly (ones with fewer than about D^2 constraints). With the violation test available, one can solve a problem with n constraints and combinatorial dimension D simply by a brute force testing of each at most d-element subset of

H (a potential basis) for optimality. This needs at most $n\left(\binom{n}{D}+\cdots+\binom{n}{0}\right) = n.O(n/D+1)^D$ violation tests. For specific applications, however, one may use a more sophisticated algorithm for these small subproblems (Clarkson suggest to use the simplex algorithm for linear programming, or the subexponential randomized algorithms of [MSW92], [Gär92] can be used for some problems).

For a deterministic counterpart of Clarkson's algorithm, we will need a stronger assumption. To every LP-type problem (H,w), we associate a set system $(H,\mathcal{R})$. For every basis $B \subseteq H$, we let $V(B)$ be the set of constraints of H violating B, and we let $\mathcal{R} = \mathcal{R}(H,w)$ be the set of all sets of the form $V(B)$ for some basis B.

We will need

> **Computational assumption 2.** A subsystem oracle for $(H,\mathcal{R})$ of dimension $\tilde{D}$ (see Definition 2.2) is available[5].

We will postpone the discussion of the subsystem oracle for linear programming to the next section.

In general, the dimension $\tilde{D}$ need not be identical to D, the combinatorial dimension of the considered LP-type problem. Typically it will be equal to D or larger. The following example shows that it can be arbitrarily large even for $D = 2$.

Example 3.1 *There exist (quite natural) LP-type problems (H,w) of combinatorial dimension 2, for which the associated set system $(H,\mathcal{R})$ has arbitrarily large VC-dimension.*

Proof sketch: We let H be a suitable finite collection of convex continuous functions from $[0,1]$ to real numbers, which are nonconstant at every subinterval of $[0,1]$. For $G \subseteq H$, we define

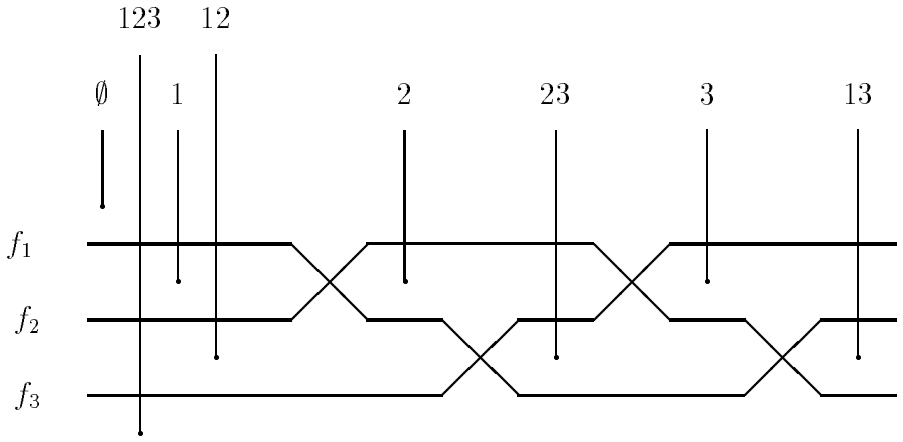
$$w(G) = \min_{x\in[0,1]} \max_{g\in G} g(x)\,.$$

By compactness and the nonconstancy assumption, for every (nonempty) G the minimum exists and it is realized in exactly one point of $[0,1]$. It is easy to verify Axioms 1 and 2 and see that the combinatorial dimension is (at most) 2. We leave it to the reader to check that for a suitably chosen H, the VC-dimension of $(H,\mathcal{R})$ can be arbitrarily large. Fig. 2 gives a hint how to shatter a 3-element set of constraints f_1, f_2, f_3 by $\mathcal{R}$ (the black points correspond to minima for certain bases; the functions belonging to those bases are not shown). □

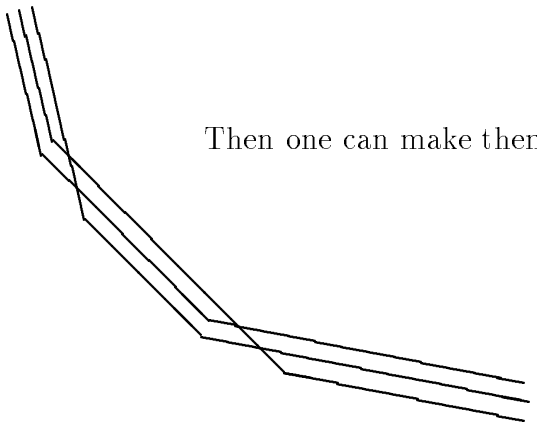
Since the randomized version of Clarkson's algorithm works with Axioms 1 and 2 and Computational assumption 1 only ([Wel92]), the assumptions needed for the randomized version are strictly weaker than we need for the deterministic case.

For simplicity, let us assume $D = \tilde{D}$ in the subsequent algorithm and analysis. If is is not the case, then the resulting estimate will hold with $\max(D,\tilde{D})$ standing

[5]Actually it suffices to have a subsystem oracle of dimension $\tilde{D}$ for any system $(H,\mathcal{R}')$ with $\mathcal{R} \subseteq \mathcal{R}'$; this is what we will in fact have in applications



It is easy for nonconvex functions.



Then one can make them convex.

Figure 2: Constructing 3 convex functions which are shattered by $\mathcal{R}$.

for D. In applications, we usually have $\tilde{D} \geq D$, and the savings in the estimate gained by distinguishing D and $\tilde{D}$ are negligible.

Now we can formulate a deterministic version of Clarkson's algorithm (a familiarity with the original Clarkson's algorithm may be helpful for understanding it). We formulate it as a recursive procedure `DetLp`, which takes a single argument $G \subseteq H$, a set of constraints, and outputs a basis for G. The procedure works as follows (in the description, we use the notation $n = |G|$, $D = \dim(H, w)$):

Procedure `DetLp`(G)

Step 1. If $n \leq n_0 = CD^4 \log D$ (C a suitable constant), compute the solution directly, by inspecting all possible at most D-element subsets of G (or by some other algorithm if available for the specific problem). Otherwise continue by the next step.

Step 2. Compute S, a $(1/r)$-net for $(G, \mathcal{R}|_G)$, where $\mathcal{R}$ is as defined above and $r = 4D^2$.

Step 3. Set $W_1 := \emptyset$, $G_1 := G$. For $i = 1, 2, \ldots$, perform the next step, until a solution is found (it will happen for $i = D$ at latest).

Step 4. Let B_i be a basis for $W_i \cup S$, computed by a recursive call of `DetLp`$(W_i \cup S)$. Let $V_i = V(B_i)$ be the set of all constraints $h \in G_i$ violating B_i. If $V_i = \emptyset$, then B_i is a basis for G and the computation finishes, otherwise set $G_{i+1} = G_i \setminus V_i$, $W_{i+1} := W_i \cup V_i$, increment i and repeat this step.

Let us first prove correctness (following Clarkson [Cla88]). For every i we have $G_i \cup W_i = G$, so the returned basis indeed defines an optimum for G. The crucial observation is that if $V_{i+1} \neq \emptyset$, then for every basis B for G V_{i+1} must contain at least one constraint of $B \setminus W_i$. Indeed, if it is not the case, we have $w(G \setminus \{h\}) = w(G)$ for every $h \in V_{i+1}$, and by a repeated application of Axiom 2 we find that $w(G \setminus V_{i+1}) = w(G)$. Also, since none of the constraints of $G \setminus V_{i+1}$ violates B_i, $w(B_i) = w(G \setminus V_{i+1}) = w(G)$, thus B_i defines an optimum of G. Since any basis of G has at most D elements, the algorithm must terminate after at most D repetitions of Step 4.

Let us analyze the running time. The fact that S is chosen as a $(1/r)$-net for $(G, \mathcal{R})$ implies that any basis violated by more than n/r constraints of G is also violated by a constraint of S. Since no constraint of S is violated by B_i, it follows that $|V_{i+1}| \leq n/r$, hence $|W_i| \leq \frac{in}{r} < n/4D$.

Let $T(n)$ denote the worst-case running time of the algorithm for n constraints. For Step 1, we get that the brute force search for solution requires at most $O(n_0/D + 1)^D = O(D)^{3D} \log^D D$ violation tests. We charge every violation test a unit cost; a more refined analysis counting violation tests separately is straightforward.

Step 2 consumes no more than $[O(D)^{7D} \log^D D]n$ time, by Theorem 2.3. Then, in each of the at most D recursive calls, we have a subproblem with at most $|S| + |W_i| = O(D^3 \log D) + \frac{n}{4D}$ constraints. If the constant C in the definition of n_0 is chosen

large enough, we get that this quantity does not exceed $n/2D$. The total cost of the violation tests is $O(Dn)$, which is negligible compared to the previously discussed overhead. We thus get the following bounds for $T(n)$:

$$\begin{array}{rcll} T(n) & \leq & O(D)^{3D}\log^D D & \text{for } n \leq n_0, \\ T(n) & \leq & [O(D)^{7D}\log^D D]n + DT(n/2D) & \text{for } n > n_0. \end{array}$$

This recurrence yields $T(n) \leq [O(D)^{7D}\log^D D]n$.

We have thus proved the following

Theorem 3.2 *Let (H, w) be a LP-type problem with n constraints of combinatorial dimension at most D, satisfying Computational assumptions 1 and 2, with $\hat{D} \leq D$. Then the optimum of (H, w) can be found deterministically in time at most $C(D)n$, where $C(D) = O(D)^{7D}\log^D D$.* □

Remark. A reader familiar with Clarkson's work may know that Clarkson has proposed also another variant of his algorithm, one with smaller sample size and "reweighting" of the constraints. We could also derandomize this variant, but it seems that this brings no improvement, since the main overhead comes from the deterministic computation of the sample, and this remains roughly the same for both methods.

4 Sample of applications

In this section we give few examples of application of Theorem 3.2 to specific geometric optimization problems. First we finish the discussion of

Linear programming. It remains to construct the subsystem oracle. A constraint h violates a basis B iff its bounding hyperplane lies above the vertex x defined by B, that is, if the bounding hyperplane intersects the open vertical semiline emanating from x upwards[6]. Thus, any set of $\mathcal{R}$ corresponds to a set of bounding hyperplanes intersecting certain vertical semiline.

For a set A of constraints, let $\bar{A}$ be the bounding hyperplanes, and consider the arragnement of $\bar{A}$. Then the semilines with endpoint within the same cell (and lower-dimensional faces bounding that cell from below) give rise to the same subsets. At this point it is convenient to use the simplifying assumptions (i)–(iii) from Section 3. Since we only look for the lowest *nonnegative* admissible vertex, it suffices to consider the portion of the arrangement of $\bar{A}$ in the nonnegative orthant. Then each cell has at least one bottommost vertex, which is defined by some k hyperplanes of $\bar{A}$ and $d-k$ of the coordinate hyperplanes bounding the nonnegative orthant. All such vertices and the sets of hyperplanes lying above them can be inspected, in at most $O(d^3+md)\left(\binom{m}{d}+\cdots+\binom{m}{0}\right)$ time ($m = |A|$), and each set of $\mathcal{R}|_A$ occurs as the sets of hyperplanes lying above some such vertex. Hence a subsystem oracle of dimension d is available and we conclude that

[6]Constraints with vertical bounding hyperplanes formally require a special treatment; this is easy to handle.

Theorem 4.1 *The linear programming problem with n constraints in R^d can be solved in $d^{7d+o(d)}n$ deterministic time.* □

Extremal ellipsoids. The *smallest enclosing ellipsoid problem* is the following: Given an n-point set P in R^d, find the smallest volume ellipsoid containing P (also called *minimum spanning ellipsoid*, *Löwner-John* ellipsoid). We have chosen this example because there is an extensive literature concerning it, and it has been considered in several recent papers related to our theme ([Pos84], [Wel91], [SW92], [Dye92]). Here the points of P play the role of the constraints, and the function w is the volume of the smallest ellipsoid enclosing a given subset. Axiom 1 is satisfied obviously, Axiom 2 follows easily from the well-known uniqueness of the Löwner-John ellipsoid [DLL57]. The combinatorial dimension is $D = (d+3)d/2$ (this is the number of degrees of freedom of an ellipsoid, see [DLL57], [Juh90], [Wel91], [SW92]).

The violation test in this case is somewhat more problematic. Specifically, it means the following: Given a set $B = \{b_1, b_2, \ldots, b_r\}$ of $r \leq D$ points in R^d and an extra point h, decide whether h is contained in the unique minimal ellipsoid containing B. The minimum enclosing ellipsoid is determined by a system of nonlinear inequalities. Post [Pos84] mentions that explicit formulae for solution can be given for $d = 2$. However, for a general dimension, no better algorithm is known to us than to apply general methods for solving systems of polynomial inequalities.

The set of points x of an ellipsoid $\mathcal{E}$ in R^d can be described by the inequality

$$(x-c)Q(x-c)^T \leq 1, \tag{3}$$

where $c \in R^d$ is the center of the ellipsoid and Q is a symmetric positive definite $d \times d$ matrix. Juhnke [Juh90] formulates necessary and sufficient conditions for such an ellipsoid $\mathcal{E}$ to be the smallest enclosing ellipsoid of B: this is iff there exist real numbers $\lambda_0 \geq 0$, $\lambda_1 > 0, \ldots, \lambda_r > 0$ such that

$$\begin{aligned} (b_j - c)Q(b_j - c)^T &= 1 \qquad\qquad j = 1, \ldots, r \\ \textstyle\sum_{j=1}^r \lambda_j b_j &= \left(\textstyle\sum_{j=1}^r \lambda_j\right) c \\ (\lambda_0 \det Q)E &= \textstyle\sum_{j=1}^r \lambda_j Q(b_j - c)^T(b_j - c)\,. \end{aligned}$$

(E denotes the unit matrix). Then the membership of another point h in this ellipsoid is expressed using (3). Hence the violation test reduces to deciding the solvability of a system of $2r + D + 1$ polynomial equalities and inequalities in $D + r$ variables (the unknowns are the entries of Q, of c and the λ_j's) of maximum degree $d + 1$. Renegar [Ren92] shows that the solvability of a system of m inequalities of maximum degree d in D variables can be decided in $(md)^{O(D)}$ time[7]. Hence a violation test can be performed in $D^{O(D)}$ time. It would be interesting to develop some more efficient methods for determining the minimum enclosing ellipsoid for $\leq D$ points (this is also a bottleneck for the randomized algorithms).

It remains to discuss the subsystem oracle. Here perhaps the easiest way is to use a "lifting transform" (pioneered by Yao and Yao [YY85] for problems of this flavor).

[7] This assumes the real RAM model of computation.

The left hand side of the inequality (3) describing an allipsoid is quadratic in the x_i's, but we can "linearize" it by mapping the problem into a higher dimensional space. Given a point $x = (x_1, \ldots, x_d) \in R^d$, we map it to a point $\varphi(x)$ in R^D, given by

$$\varphi(x) = (x_1, \ldots, x_d, x_1^2, x_1x_2, x_1x_3, \ldots, x_1x_d, x_2^2, x_2x_3, \ldots, x_d^2)\,.$$

Then the condition $x \in \mathcal{E}$ or (3) is equivalent to $\varphi(x) \in h$, where $h = h(Q, c)$ is certain halfspace in R^D, determined by Q and c.

For a given $A \subseteq P$, we want to find all sets definable by ellipsoids determined by bases of P. Being somewhat generous, we include subsets of A definable by *all* ellipsoids. In fact, we map A into R^D by φ, and we list the preimages of all subsets of $\varphi(A)$ definable by halfspaces in R^D. The number of such subsets is still $O(|A|)^D$, and we can list them in $O(|A|)^{D+1}$ time (using essentially the method discussed above for linear programming). Hence a subsystem oracle of dimension D is available and we can apply the general result. Let us remark that, strictly speaking, the algorithm only computes the minimal subset determining the ellipsoid; the ellipsoid itself is given implicitly, as a solution of the above system of inequalities and equalities.

A problem of very similar flavor is finding the maximum volume ellipsoid inscribed into the intersection of n given halfspaces in R^d. This ellipsoid is again unique [DLL57], and the combinatorial dimension is again D. Both violation test and subsystem oracle can be handled in a similar way as for the previous problem. We get

Theorem 4.2 *The minimum volume enclosing ellipsoid for a set of n points in R^d or the maximum volume ellipsoid inscribed into the intersection of n halfspaces in R^d can be computed deterministically in $D^{O(D)}n$ time, $D = d(d+3)/2$. The computation needs $D^{7D+o(D)}n$ arithmetic operations plus $D^{3D+o(D)}n$ violation tests.*

We believe that the above examples sufficiently illustrate the technique of applying the general result to specific geometric optimization problems. In general, problems similar to convex programming involving bounded degree polynomials should be amenable to such treatment. It would be interesting to find also nongeometric applications for the Sharir-Welzl framework and the algorithms.

References

[BR91] B. Berger and J. Rompel. Simulating $(\log n)^c$-wise independence in NC. *Journal of the ACM*, 38(4):1028–1046, 1991.

[BRS89] B. Berger, J. Rompel, and P. W. Shor. Efficient NC algorithms for set cover with applications to learning and geometry. In *Proc. 30. IEEE Symposium on Foundations of Computer Science*, pages 54–59, 1989.

[CF90] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[Cha91] B. Chazelle. Cutting hyperplanes for divide-and-conquer. Tech. report CS-TR-335-91, Princeton University, 1991. Preliminary version: *Proc. 32. IEEE Symposium on Foundations of Computer Science*, October 1991.

[Cla86] K. Clarkson. Linear programming in $O(n \times 3^{d^2})$ time. *Information Processing Letters*, 22(1):21–24, 1986.

[Cla88] K. Clarkson. Las Vegas algorithm for linear programming when the dimension is small. In *Proc. 29. IEEE Symposium on Foundations of Computer Science*, pages 452–457, 1988.

[DF87] M. E. Dyer and A. M. Frieze. A randomized algorithm for fixed-dimensional linear programming. Manuscript, 1987.

[DLL57] L. Danzer, D. Laugwitz, and H. Lenz. Über das Löwnersche Ellipsoid und seine Anlagen unter den einem Eikörper einbeschriebenen Ellipsoid. *Archiv der Mathematik*, 8:214–219, 1957.

[Dye86] M. E. Dyer. On a multidimensional search technique and its application to the Euclidean 1-centre problem. *SIAM Journal on Computing*, 15:725–738, 1986.

[Dye92] M. E. Dyer. A class of convex programs with applications to computational geometry. In *Proc. 8. ACM Symposium on Computational Geometry*, pages 9–15, 1992.

[Gär92] B. Gärtner. Abstract optimization problems. In *Proc. 33. IEEE Symposium on Foundations of Computer Science*, 1992. To appear.

[HW87] D. Haussler and E. Welzl. ε-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987.

[Juh90] F. Juhnke. Volumenminimale Ellipsoidüberdeckungen. *Beiträge zur Algebra und Geometrie*, 30:143–153, 1990.

[Kal92] G. Kalai. A subexponential randomized simplex algorithm. In *Proc. 24. ACM Symposium on Theory of Computing*, 1992.

[Kar84] N. Karmakar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[Kha80] L. G. Khachiyan. Polynomial algorithm in linear programming. *U.S.S.R. Comput. Math. and Math. Phys.*, 20:53–72, 1980.

[KPW92] J. Komlós, J. Pach, and G. Wöginger. Almost tight bounds for epsilon-nets. *Discrete & Computational Geometry*, 1992. To appear.

[Mat91] J. Matoušek. Approximations and optimal geometric divide-and-conquer. In *Proc. 23. ACM Symposium on Theory of Computing*, pages 506–511, 1991.

[Meg84] N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31:114–127, 1984.

[MNN89] R. Motwani, J. Naor, and M. Naor. The probabilistic method yields deterministic parallel algorithms. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 8–13, 1989.

[MSW92] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. In *Proc. 8. ACM Symposium on Computational Geometry*, pages 1–8, 1992.

[MWW92] J. Matoušek, E. Welzl, and L. Wernisch. Discrepancy and ε-approximations for bounded VC-dimension. *Combinatorica*, 1992. To appear; also in Proc. 32. IEEE Symposium on Foundations of Computer Science (1991), pages 424–430.

[Pos84] M. J. Post. Minimum spanning ellipsoids. In *Proc. 16. ACM Symposium on Theory of Computing*, pages 108–116, 1984.

[Ren92] J. Renegar. On the computational complexity and geometry of the first order theory of the reals. *Journal of Symbolic Computation*, 1992.

[Sau72] N. Sauer. On the density of families of sets. *Journal of Combinatorial Theory Ser. A*, 13:145–147, 1972.

[Sei91] R. Seidel. Small dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6(5):423–434, 1991.

[Spe87] J. Spencer. *Ten lectures on the probabilistic method.* CBMS-NSF, SIAM, 1987.

[SW92] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proc. 1992 Symposium on Theoretical Aspects of Computer Science (Lecture Notes in Computer Science)*, volume 577, pages 569–579. Springer-Verlag, 1992.

[VC71] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.

[Wel91] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *LNCS 555 (New Results and New Trends in Computer Science)*, pages 359–370. Springer-Verlag, 1991.

[Wel92] E. Welzl. New results on linear programming and related problems (survey paper). Manuscript, 1992.

[YY85] F. F. Yao and A. C. Yao. A general approach to geometric queries. In *Proc. 17. ACM Symposium on Theory of Computing*, pages 163–168, 1985.