APPEARED IN Proc. 13th Ann. Symp. on Theoret. Aspects of Comput. Sci.
Lecture Notes in Computer Science **1046** (1996) 669-687.

Linear Programming – Randomization and Abstract Frameworks*

Bernd Gärtner and Emo Welzl
Institut für Informatik, Freie Universität Berlin
Takustr. 9, D-14195 Berlin, Germany
{gaertner,emo}@inf.fu-berlin.de

Abstract

Recent years have brought some progress in the knowledge of the complexity of linear programming in the *unit cost model*, and the best result known at this point is a *randomized 'combinatorial'* algorithm which solves a linear program over d variables and n constraints with expected

$$O(d^2n + e^{O(\sqrt{d \log d})})$$

arithmetic operations. The bound relies on two algorithms by Clarkson, and the subexponential algorithms due to Kalai, and to Matoušek, Sharir & Welzl.

We review some of the recent algorithms with their analyses. We also present abstract frameworks like *LP-type problems* and *abstract optimization problems* (due to Gärtner) which allow the application of these algorithms to a number of non-linear optimization problems (like polytope distance and smallest enclosing ball of points).

1 Introduction.

We consider the problem of minimizing some linear objective function in d nonnegative variables subject to n linear inequalities. In geometric terms, this corresponds to finding a point extremal in some direction inside a polyhedron defined as the intersection of n halfspaces and the positive orthant in d-space, see e.g. [6, 25]. There is vivid interest in this problem, and two major questions have dominated the picture for a long time. First, are linear programming problems solvable in polynomial time? Second, is the simplex method for solving linear programs a polynomial algorithm? The first question was answered in the affirmative by Khachiyan [18] who developed a polynomial time algorithm in the bit-model (Turing machine model), its complexity depending on the encoding size of the input. The second question is still open, not only for the simplex method but for any 'combinatorial' algorithm whose runtime has a bound in the unit cost model (RAM model), i.e. depends only on n and d. For many pivot rules, the simplex method was shown to require exponential time on certain inputs (the first such input has been constructed by Klee and Minty [19] for the pivot rule originally proposed by Dantzig [9]). Two lines of research have been pursued in order to overcome the exponential worst case behavior. First, the *average* complexity has been studied which assumes that the input is random with respect to some probability distribution. In this model, a polynomial bound for the simplex method (under a particular pivot rule) has first been obtained by Borgwardt [4], partially explaining the good performance of the method on practical problems. Second, the *randomized* complexity has been investigated, which replaces the average over an input distribution by the average over internal 'coin flips' performed by a

*supported by a Leibniz Award from the German Research Society (DFG), We 1265/5-1.

randomized algorithm. Although the progress obtained in this direction does not match the one achieved for the average complexity in the sense that polynomial bounds could not be shown, at least the exponential worst case behavior has been beaten, and the best result known at this point is a randomized combinatorial algorithm which computes the solution to a linear program with an expected *subexponential* number of $O(d^2n + e^{O(\sqrt{d \log d})})$ arithmetic operations. The bound relies on two algorithms by Clarkson [8], and the subexponential algorithms due to Kalai [17], and to Matoušek, Sharir, and Welzl [20].

The subexponential bound is the last link so far in a chain of results that have brought about considerable progress over the last decade. Here is a brief account of the history of LP in the unit cost model. The runtime of the simplex method is bounded by the number of vertices of an n-facet polyhedron, which is $O(n^{\lfloor d/2 \rfloor})$ (ignoring an extra factor for the pivot steps). Megiddo [23] has given the first algorithm whose runtime is of the form $O(C_d n)$, and is thus linear in n for any fixed d (see also [11] for the cases $d = 2, 3$). The factor C_d, however, is 2^{2^d}. An improvement to $C_d = 3^{d^2}$ was given in [12] and [7]. Randomized algorithms were then suggested by Dyer and Frieze [13], achieving $C_d = d^{3d}$, and Clarkson [8] with a complexity of $O(d^2n + d^4\sqrt{n} \log n + d^{d/2+O(1)} \log n)$. Later, Seidel [24] discovered a remarkably simple randomized LP algorithm with expected time $O(d!n)$, [26] provide a bound of $O(d^3 2^d n)$. Kalai published the first subexponential bound in [17], and then a similar bound was proved in [20]. Combining those subexponential bounds with Clarkson's algorithms in [8] gives the above mentioned currently best bound of $O(d^2n+e^{O(\sqrt{d \log d})})$ for randomized algorithms. The best deterministic bound is $O(d^{O(d)}n)$ which has been obtained by Chazelle and Matoušek [5] via derandomization of one of Clarkson's randomized procedures.

In this survey we review the algorithms and the analyses which lead to the expected time bound claimed in the abstract. We restrict ourselves to a presentation of the subexponential algorithm in [20] and do not explicitly review Kalai's subexponential simplex algorithms [17]. However, as pointed out by Goldwasser [16], one of his variants is exactly dual to an algorithm we give in Section 3; moreover, ideas underlying another variant are incorporated into an algorithm presented in Section 5, so Kalai's contribution is implicitly contained in this paper.

One nice feature of the combinatorial algorithms presented below is that they can be formulated in a quite general abstract framework, and so they are applicable to a number of nonlinear optimization problems,[1] as e.g. computing the smallest ball (ellipsoid) enclosing n points in d-space, computing the largest ellipsoid in a convex d-polytope with n facets, computing the distance between two convex d-polytopes with n facets or with n vertices, rectilinear 2- and 3-center problem etc. Similar bounds as the one claimed above hold for all these problems, provided one can perform certain primitive operations efficiently. This has been shown by Gärtner [14] who also gave explicit primitives for the polytope distance and the minimum spanning ball problem [15]. We will elaborate on abstract frameworks underlying those extensions in the second half of the paper.

2 Notation and Basics

For parameters $n, d > 0$, a linear programming problem in standard form consists of finding a *nonnegative* vector $x \in \mathbb{R}^d$ that minimizes a linear function $c^T x$ subject to n linear inequalities $\sum_{j=1}^d a_{ij}x_j \leq b_i$, $i = 1, \ldots, n$. In compact form this can be written as

$$\begin{array}{lll} \text{(LP)} & \text{minimize} & c^T x \\ & \text{subject to} & Ax \leq b, \\ & & x \geq 0, \end{array} \tag{1}$$

where c is a d-vector, b is an n-vector and A is an $(n \times d)$-matrix. The $n+d$ inequalities $Ax \leq b$ and $x \geq 0$ are the *constraints* of the linear program. (Due to their special nature, the latter are called *nonnegativity constraints*.) The set of vectors x satisfying all the constraints is called the *feasible region* of the LP, and this region is an intersection of halfspaces, hence a polyhedron in $\mathbb{R}^d$ whose

[1] An extension of Clarkson's algorithm to convex programming has been worked out by Adler and Shamir [1].

facets are induced by some of the *constraint hyperplanes*. If the polyhedron is nonempty, the LP is called *feasible*. In geometric terms, linear programming is therefore the problem of finding a point in a polyhedron that is extreme in a given direction c.

To avoid various technicalities, we assume that the cost vector c satisfies $c \geq 0$. This ensures that (1) is *bounded*, equivalently that $c^T x$ assumes a (nonnegative) minimum value in the feasible region, provided it is nonempty.

Let H be the set of n halfspaces defined by the constraints $Ax \leq b$, and let H_+ denote the set of d halfspaces defined by $x \geq 0$. For $G \subseteq H \cup H_+$, we let v_G denote the (unique) *lexicographically minimal* point x minimizing $c^T x$ over $P_G := \bigcap_{h \in G} h$, the intersection of halfspaces in G. At least for $H_+ \subseteq G$, v_G exists (by our assumption on c), and $v_{H_+} = (0, \ldots, 0)^T$. If P_G is empty, we let $v_G := \infty$, which denotes *infeasibility*. If P_G is nonempty but no lexicographically smallest point minimizing $c^T x$ exists, we let $v_G := -\infty$, standing for *unboundedness*. If (1) is feasible, $v_{H \cup H_+}$ is its unique *optimal solution*. Let us write $v_F < v_G$ if $c^T v_F < c^T v_G$, or if $c^T v_F = c^T v_G$ but v_F is lexicographically smaller than v_G. Moreover, $-\infty$ precedes (and ∞ succeeds) any finite vertex in this ordering.

A set of constraints (halfspaces) $B \subseteq H \cup H_+$ is called a *basis* if v_B is finite but $v_{B'} < v_B$ for any proper subset B' of B. For example, H_+ is a basis. If v_G is finite, then a *basis of* G is a minimal subset $B \subseteq G$ with $v_B = v_G$. A constraint $h \in H \cup H_+$ is *violated* by G if and only if $v_G < v_{G \cup \{h\}}$. Uniqueness of v_G implies that this is equivalent to $v_G \notin h$ (if v_G is finite), which explains the intuitive term 'violation'. Finally, h is *extreme* in G if $v_{G - \{h\}} < v_G$, i.e. if h is violated by $G - \{h\}$.

With this terminology we can state a first easy – but important – lemma. Part (i) and (ii) are obvious, while (iii) is standard in linear programming theory.

Lemma 1

(i) For $F \subseteq G \subseteq H \cup H_+$, $v_F \leq v_G$.

(ii) If v_F, v_G are finite with $v_F = v_G$, then h is violated by F if and only if h is violated by G.

(iii) If v_G is finite, then any basis of G has exactly d constraints, and G has at most d extreme constraints.

We remark that our scenario here is dual to a standard setup for the simplex method. The assumption $c \geq 0$ corresponds to the *feasible origin* assumption, while our convention of resolving ambiguities by distinguishing the lexicographically smallest optimal solution is the simplex analogue of resolving degeneracies by symbolic perturbation. While the feasible origin assumption is for technical convenience only and can be removed if necessary, symbolic perturbation is one way of achieving correctness of the simplex method (see [6] for details). Exactly the same relations hold between our setup and the algorithms we are going to develop below.

3 Algorithms

We describe three algorithms which in combination prove the claimed $O(d^2 n + e^{O(\sqrt{d \log d})})$ bound. Clarkson's first algorithm `cl1_lp` solves 'large' problems ('large' means that n is large compared to d). It uses Clarkson's second algorithm `cl2_lp` for problems of size roughly $d\sqrt{n}$. This algorithm invokes the subexponential algorithm `subex_lp`, when the number of constraints is at most $6d^2$.

In this section we will not explicitly refer to the vector c (it is hidden in the v_G-notation). Moreover, the linear program we consider is assumed to be feasible (although the reader will realize that this is not essential, if we let the algorithms stop as soon as infeasibility is discovered). We use v_G^+ short for $v_{G \cup H_+}$. Given some set H of constraints, the goal of the algorithms is to compute v_H^+.

Clarkson's algorithm 1. In order to solve the problem for a set H of n constraints we choose some sample R of $d\sqrt{n}$ constraints, compute $v = v_R^+$ (with some other algorithm), and determine the set V of constraints in H violated by v. If V is not too large, ($|V| \le 2\sqrt{n}$), we add V to some initially empty set G, choose another sample R, compute $v = v_{G\cup R}^+$, add the violated constraints (provided there are not too many) to G, and so on. We continue until no constraint is violated by v, when the solution is found. A more concrete specification of the procedure is given below ($\binom{H}{r}$ denotes the family of r-elements subsets of H, the random R chooses each of them with the same probability):

```
function procedure cl1_lp(H)                H: set of n constraints in d-space
    if n ≤ 9d² then                                                  returns v_H^+
        return cl2_lp(H)
    else
        r := ⌊d√n⌋;  G := ∅
        repeat
            choose random R ∈ (H choose r)
            v := cl2_lp(G ∪ R)
            V := {h ∈ H | v violates h}
            if |V| ≤ 2√n then G := G ∪ V
        until V = ∅
        return v
```

There are two crucial facts which make this procedure efficient. First, the expected size of V is no more than $\sqrt{n}$ (proved below); thus, the probability that $|V| > 2\sqrt{n}$ is at most $\frac{1}{2}$ (by Markov's inequality), and the expected number of attempts to find a sufficiently small V is at most 2. Second, if any constraint is violated by $v = v_{G\cup R}^+$, then for each basis B of $H \cup H_+$, there must be a constraint $h \in B - (G \cup R \cup H_+)$ which is violated by v. (If no constraint in B is violated by v then $v_H^+ = v_B^+ \le v_{G\cup R\cup B}^+ = v_{G\cup R}^+ \le v_H^+$, and v is already the solution of H.) It follows that we augment G at most d times. Summing up, the expected number of iterations through the `repeat`-loop is bounded by $2d$. We conclude:

Lemma 2 *For $n = |H| > 9d^2$, procedure* `cl1_lp` *computes v_H^+ with an expected number of $O(d^2 n)$ arithmetic operations, and an expected number of at most $2d$ calls to* `cl2_lp` *with at most $3d\sqrt{n}$ constraints.* □

A sampling lemma. It remains to prove the bound on the expected size of the sets V of violated constraints. For further application, we formulate (and prove) the lemma for multisets.

Lemma 3 *Let G be a set of constraints in d-space, let H be a multiset of n constraints in d-space, and let $1 \le r \le n$. Then, for random $R \in \binom{H}{r}$, the expected size of $V_R := \{h \in H \mid v_{G\cup R}^+ \text{ violates } h\}$ is bounded by $d\frac{n-r}{r+1}$.*

Proof. For $R \in \binom{H}{r}$ and $h \in H$, let $\chi_G(R, h)$ denote the characteristic function for the event that $v_{G\cup R}^+$ violates h (one if true, and zero otherwise). Then

$$\binom{n}{r}\mathrm{E}(|V_R|) = \sum_{R\in\binom{H}{r}} \sum_{h\in H-R} \chi_G(R,h) =$$

$$\sum_{Q\in\binom{H}{r+1}} \sum_{h\in Q} \chi_G(Q-\{h\},h) \le \sum_{Q\in\binom{H}{r+1}} d = d\binom{n}{r+1};$$

the first equality is a write-out of the definition, the second equality is a rewriting of sums, and the inequality follows from the fact that there are at most d extreme constraints in every set $G\cup Q\cup H_+$; (in fact, only those extreme constraints which are in Q contribute to the sum). $\mathrm{E}(|V_R|) \le d\frac{n-r}{r+1}$ follows. □

Hence, for $r = \lfloor d\sqrt{n}\rfloor$, we have $\mathrm{E}(|V_R|) < \sqrt{n}$. Below, we will use the lemma for multisets and with $r = 6d^2$ which entails $\mathrm{E}(|V_R|) < \frac{1}{6d}n$.

Clarkson's algorithm 2. This algorithm proceeds very similar to the first one. It chooses a random sample R of constraints – now of size $6d^2$ – and computes v_R^+ by some other algorithm (the subexponential algorithm to be described later). Then it determines the violated constraints V from H. Instead of 'forcing' these constraints for the next iterations (like in the previous algorithm), it just increases their probability to be selected in further random samplings on H. This effect is achieved by assigning a multiplicity μ_h to every constraint h in H; μ_h is doubled, when h is violated by an intermediate solution. The analysis will show, that for any basis B of H, the elements of B increase their multiplicities so quickly, that they are chosen with high probability after a logarithmic number of iterations. Let us first provide the details of the procedure, and then have the analysis materialize. In this procedure, we view H as a multiset, each h with multiplicity μ_h; $\mu(H) := \sum_{h \in H} \mu_h$. When we talk about random $R \in \binom{H}{r}$, then we refer also to the multiset (and so, in general, R is a multiset).

```
function procedure cl2_lp(H)              H: set of n constraints in d-space
    if n ≤ 6d² then                                     μ_h = 1 for all h ∈ H
        return subex_lp(H)                                        returns v_H^+
    else
        r = 6d²
        repeat
            choose random R ∈ (H choose r)
            v := subex_lp(R)
            V := {h ∈ H | v violates h}
            if μ(V) ≤ (1/3d) μ(H) then
                for all h ∈ V do μ_h := 2μ_h
        until V = ∅
        return v
```

It follows from Lemma 3 and Markov's inequality that the expected number of attempts until we obtain a sufficiently small V is 2 at most. The next lemma will bound the number of successful iterations, i.e. iterations where H gets reweighted.

Lemma 4 *Let k be some positive integer. After kd successful iterations, we have*

$$2^k \leq \mu(B) < ne^{k/3}$$

for every basis B of H.

Proof. Every successful iteration adds a weight of at most $\frac{1}{3d}\mu(H)$ to H, and so the upper bound

$$\mu(B) \leq \mu(H) \leq n(1 + \frac{1}{3d})^{kd} < ne^{k/3}$$

follows.

To see the lower bound on $\mu(B)$, recall that (as we argued before) if there are constraints violated by v, then there must be a violated constraint in any basis of H. So every successful iteration doubles the weight of a constraint in B. That is, there is a constraint in B which has been doubled at least k times (since there are only d constraints in B). Hence, $\mu(B) \geq 2^k$. ◻

Since $2 > e^{1/3}$, the lower bound will exceed the upper bound for large enough k, which means that we must have found the solution before that.

Lemma 5 *For $n = |H| > 6d^2$, procedure* `cl2_lp` *computes v_H with an expected number of $O(d^2 n \log n)$ arithmetic operations, and an expected number of at most $6d \ln n$ calls to* `subex_lp` *with at most $6d^2$ constraints.*

Proof. For $k = 3 \ln n$, we have $2^k = n^{3 \ln 2} > n^2 = ne^{k/3}$. Hence, by Lemma 4, there are at most $3d \ln n$ successful iterations, and the expected number of iterations is at most twice this number. Each iteration costs $O(dn)$ arithmetic operations and one call to `subex_lp`. ◻

A subexponential algorithm. Given a set H of n constraints we remove a random constraint h, and compute the solution for $H - \{h\}$ recursively (so after taking random samples of size $d\sqrt{n}$ and $6d^2$, we ended up with a sample of size $n-1$.) If h is not violated by $v^+_{H-\{h\}}$, then we are done. If h is violated, then we try again by removing a (hopefully different) random constraint. Note that the probability that h is violated by $v^+_{H-\{h\}}$ is at most $\frac{d}{n}$ since there are at most d extreme constraints in H (actually a special case of Lemma 3).

So much for the basic idea; in order to guarantee efficiency, we need some additional ingredients. First, the procedure `SUBEX_lp` actually solving the problem has two parameters: a set G of constraints, and a basis $B \subseteq G$; in general, B is *not* a basis *of* G, and we call it a *candidate basis.* [2] For technical reasons the subexponential procedure computes v_G (rather than v^+_G) and a basis for G; it assumes that $v_B > -\infty$ and so $v_G > -\infty$. Note that B has no influence on the output of the procedure (but influences its efficiency). The original problem of computing v^+_H can now be solved by:

```
function procedure subex_lp(H)            H: set of n constraints in d-space
    (v, B) := SUBEX_lp(H ∪ H+, H+)                                returns v+_H
    return v
```

Besides the violation test, the following pseudocode for `SUBEX_lp` assumes the availability of a second primitive operation, $\texttt{basis}(B, h)$, which computes a basis of $B \cup \{h\}$ for a d-element basis B and a constraint h. This step corresponds to a dual pivot step, and with an appropriate representation of B, it can be performed with $O(d^2)$ arithmetic operations. Our feasibility assumption guarantees that $\texttt{basis}(B, h) \neq \infty$. Note that the quantity m below is initially $n + d$ before we go down the recursion.

```
function procedure SUBEX_lp(G, B);                G: set of m constraints
    if G = B then                                         B ⊆ G: basis
        return (v_B, B)                       returns v_G and basis of G
    else
        choose random h ∈ G − B
        (v, B') := SUBEX_lp(G − {h}, B)
        if h violates v then
            return SUBEX_lp(G, basis(B', h))
        else
            return (v, B')
```

A simple inductive argument shows that the procedure returns the required answer. This happens after a finite number of steps, since the first recursive call decreases the number of constraints, while the second call increases the value of the candidate basis (and there are only finitely many different values). The procedure `SUBEX_lp` can be viewed as a dual simplex algorithm, its corresponding primal version being exactly one of Kalai's subexponential simplex variants [17, 16].

For the analysis of the expected behavior of the algorithm, let us take a closer look at the probability to make the second recursive call with candidate basis '$\texttt{basis}(B', h)$'. To be precise we have to take back the previously claimed $\frac{d}{n}$-bound and replace it by $\frac{d}{m-d}$, since we choose h from $G-B$ (and B always has d elements). However, if $d-j$ extreme constraints in G are in B, then the bound improves to $\frac{j}{m-d}$; in fact, there are never more 'bad' choices than there are choices at all, and so the bound can be lowered to $\frac{\min\{j,m-d\}}{m-d}$. We want to show that the numerator decreases rapidly as we go down the recursion, and this will entail the subexponential time bound.

We enumerate the constraints in G in such a way that

$$v_{G-\{h_1\}} \leq v_{G-\{h_2\}} \leq \cdots \leq v_{G-\{h_{d-k}\}} < v_B \leq v_{G-\{h_{d-k+1}\}} \leq \cdots \leq v_{G-\{h_m\}}.$$

The ordering is not unique, but the parameter k emerging from this ordering is unique and we call it the *hidden dimension* of the pair (G, B); hidden dimension zero implies that B is a basis for G.

[2] In simplex terminology, this corresponds to a dual feasible basis.

Note that for $h \in G - B$, $v_B \le v_{G-\{h\}}$, and so $h_1, h_2, \ldots, h_{d-k}$ are in B. Hence the only choices for h which possibly entail a second call are $h_{d-k+1}, \ldots, h_d$ (for $i > d$, $v_{G-\{h_i\}} = v_G$). Suppose that, indeed, h_{d-k+i} , $1 \le i \le k$, is chosen, and the first call (with candidate basis B) returns with basis B', $v_{B'} = v_{G-\{h_{d-k+i}\}}$. Then we compute $B'' = \texttt{basis}(B', h_{d-k+i})$. Since $v_{G-\{h_{d-k+i}\}} = v_{B'} < v_{B''}$, the pair (G, B'') for the second call has hidden dimension at most $k-i$.

The hidden dimension is monotone, i.e., if $B \subseteq F \subseteq G$, then the hidden dimension of (F, B) does not exceed the hidden dimension of (G, B). This holds, because the constraints $h_1, h_2, \ldots, h_{d-k}$ are in B (and so in F), and $v_{F-\{h_i\}} \le v_{G-\{h_i\}}$ because $F \subseteq G$.

We denote by $b(m, k)$ the maximum (over all possible inputs) expected number of calls to `basis` entailed by a call $\texttt{SUBEX\_lp}(G, B)$ with m constraints and hidden dimension at most k. $b(d, k) = 0$ for $0 \le k \le d$, and with the discussion above we conclude the recursion

$$b(m,k) \le b(m-1,k) + \frac{1}{m-d} \sum_{i=1}^{\min\{k,m-d\}} (1 + b(m, k-i)) \quad \text{for } m > d. \tag{2}$$

A simple proof by induction shows that $b(m, k) \le 2^k(m - d)$. However, it turns out that for n not much larger than d, this is a gross overestimate, and with extra effort one can show

$$1 + b(m,k) \le \exp\left(2\sqrt{k \,\underline{\ln}\frac{m-d}{\sqrt{k}}} + (\ln 3 + 2)\sqrt{k} + \underline{\ln}\frac{m-d}{\sqrt{k}}\right) = e^{O(\sqrt{k \ln(m-d)})},$$

where $\underline{\ln}\ x = \max(\ln x, 1)$. For a proof of this bound see [15, 20]. Each basis computation takes $O(d^2)$. For each basis B computed in the algorithm, we test every constraint at most once for violation of v_B. Hence, the number of violation tests is bounded[3] by $(m - d)b(m, d)$, with $O(d)$ arithmetic operations for each such test.

Lemma 6 *For $n = |H|$, procedure $\texttt{subex\_lp}(H)$ computes $v_H^+ = v_{H \cup H_+}$ with an expected number of $O((d^2 + nd)e^{O(\sqrt{d \ln n})})$ arithmetic operations.* □

Putting it together. If we plug the bound obtained for `subex_lp` into Lemma 5, then we have a bound of $O(d^2 n \log n + e^{O(\sqrt{d \ln d})} \log n)$ for the expected number of arithmetic operation required by `cl2_lp`. If we apply this to Lemma 2, we get an estimate of $O(d^2 n + d^4\sqrt{n} \log n + e^{O(\sqrt{d \ln d})} \log n))$ for `cl1_lp`. The middle term is dominated by the first or last term for all values of d and n; for the values of n for which the last term is dominant, the $\log n$ factor is absorbed by $O(\sqrt{d \ln d})$ in the exponent. We conclude

Theorem 7 *The optimal nonnegative vertex v_H^+ of a linear program with n constraints H in d-space can be computed by a randomized algorithm with an expected number of $O(d^2 n + e^{O(\sqrt{d \ln d})})$ arithmetic operations.* □

4 An Abstract Framework

In this section we show that the algorithms we discussed so far actually work in a more general and abstract combinatorial setting, which makes them applicable to a wide range of problems (most of them are convex programming).

Let us consider optimization problems specified by pairs (H, w), where H is a finite set, and $w : 2^H \to \mathcal{W} \cup \{-\infty\}$ is a function with values in a linearly ordered set $(\mathcal{W} \cup \{-\infty\}, \le)$, the value $-\infty$ (standing for 'undefined') preceding all values in $\mathcal{W}$. The elements of H are called *constraints*, and for $G \subseteq H$, $w(G)$ is called the *value of* G. The goal is to compute a minimal subset B_H of H with the same value as H (from which, in general, the value is easy to determine), assuming the availability of two basic operations to be specified below. It turns out that the algorithms we

[3] A direct proof also gives the $2^k(m - d)$ bound for the expected number of violation tests.

have seen can be used to perform this computational task, as long as the following axioms are satisfied:

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

If Axioms 1 and 2 hold, then we call (H, w) an *LP-type problem.* Linear programming is an LP-type problem $(H \cup H_+, w)$, if we set $w(G) = v_G$ for a constraint set $G \subseteq H \cup H_+$. Then the axioms coincide with Lemma 1 (i) and (ii). The notions introduced in Section 2 carry over in the obvious way: A *basis* B is a set of constraints with $w(B') < w(B)$ for all proper subsets B' of B. For $G \subseteq H$, a *basis of* G is a minimal subset $B = B_G$ of G with $w(B) = w(G)$. Constraint h is *violated by* G, if $w(G) < w(G \cup \{h\})$. Constraint h is *extreme in* G, if $w(G - \{h\}) < w(G)$.

For the efficiency of the algorithm the following parameter is crucial: the maximum cardinality of any basis is called the *combinatorial dimension* of (H, w), and is denoted by $\delta = \delta_{(H,w)}$.

We assume that the following primitive operations are available.

> (*Violation test*) 'h `is violated by` B', for a constraint h and a basis B, tests whether h is violated by B or not.
>
> (*Basis computation*) '`basis`(B, h)', for a constraint h and a basis B, computes a basis of $B \cup \{h\}$.

Carefully reconsidering the algorithms developed in the previous section, we find that the two axioms and primitive operations above constitute all that is needed to make them work in the abstract setting. Of course, concrete vertices v_B as referred to in case of LP do not exist in the abstract setting and need to be replaced by their abstract representations B.

As far as the subexponential analysis of `SUBEX_lp` is concerned, we have already seen that another property of LP is needed to make it valid, and this is part (iii) of Lemma 1 which states that any basis has exactly d constraints. We call this property *basis-regularity.* More precisely, it is crucial for the analysis that a call to `SUBEX_lp`(G, B) with $|G| = d$ entails no further calls to primitive operations, and this follows from basis-regularity. The axioms above do not imply basis-regularity, and all of the concrete LP-type problems we present below are indeed not basis-regular. Still, a bound which is linear in n (but exponential in δ) can be proved in any case [26, 15].

Theorem 8 *Given an LP-type problem (H, w), $|H| = n$, of combinatorial dimension δ, and some initial basis $B \subseteq H$, a basis B_H of H can be computed with an expected number of at most $2^{\delta+2}(n - \delta)$ violation tests resp. basis computations.* ◻

The next section shows that the subexponential analysis can in fact be extended to all LP-type problems. However, already at this stage, the theorem establishes optimal linear-time algorithms for any LP-type problem of fixed combinatorial dimension δ. To this end observe that the primitive operations involve problems of size $\delta + 1$ at most, and for constant δ, even brute-force realizations entail only constant effort. There is quite a number of problems which fit into the framework. We just provide a list, for further details see [20].

Smallest enclosing ball. Given a set P of n points (or other objects) in d-space, find center and radius of the smallest ball containing all the points (combinatorial dimension $d + 1$).

Polytope distance. Given two polytopes P and Q (specified by a total of n points in d-space), compute points $p \in P, q \in Q$ minimizing $\text{dist}(p, q)$, $p \in P, q \in Q$ (combinatorial dimension $d + 2$).

Smallest enclosing ellipsoid. Given n points in d-space, compute the ellipsoid of smallest volume containing the points (combinatorial dimension $d(d + 3)/2$).

Largest ellipsoid in polytope. Given a polytope in d-space as the intersection of n halfspaces, compute the ellipsoid of largest volume contained in the polytope (combinatorial dimension $d(d + 3)/2$).

Smallest intersecting ball. Given n closed convex objects in d-space, compute the ball of smallest radius that intersects all of them (combinatorial dimension $d + 1$).

Angle-optimal placement of point in polygon. Let P be a star-shaped polygon with n vertices in the plane (a polygon is *star-shaped* if there is a point inside the polygon which sees all edges and vertices; the locus of all such points is called the *kernel*). Compute a point p in the kernel of P, such that after connecting p to all the vertices of P by straight edges, the minimal angle between two adjacent edges in the triangulated polygon is maximized (combinatorial dimension 3).

Rectilinear 3-centers in the plane. Given a set P of n points in the plane, find three points c_1, c_2, c_3 so that $\max_{p \in P} \min_{i=1,2,3} \|c_i - p\|_\infty$ is minimal [27].

Spherical separability. Given n red and n blue segments in d-space, find the smallest radius ball covering all red segments and disjoint from all blue segments. Holds also for many computationally simple objects instead of line segments [28].

Width of 'thin' point sets in the plane. The width of a planar set P is the smallest width of a stripe covering P. Computing the width of an n-point set has an $\Omega(n \log n)$ lower bound. However, if the smallest distance between two points in P is larger than the width of P, the problem can be formulated as an LP-type problem of combinatorial dimension 5 (relies on the fact that n disjoint unit disks allow a line transversal iff any 5 of them do [10]).

The question whether a problem is LP-type can be a subtle issue and the answer may depend on the way the problem is formulated. In LP, for example, it may seem natural to define $w(G)$ just as the minimum value of $c^T x$ subject to the constraints in G. Then however, basis regularity and – much worse – the locality axiom no longer hold. To see this, consider the 2-dimensional problem of minimizing y (i.e. $c^T = (0, 1)$) subject to the constraint set $H = \{h_1, h_2, h_3\}$, $h_1 : x - y \leq -1$, $h_2 : x \geq 0$ and $h_3 : y \geq 0$. Then $w(H) = w(\{h_1, h_2\}) = 1$, and the latter set is a basis (of H). Another basis of different size is $\{h_3\}$ with $w(\{h_3\}) = w(\{h_1, h_3\}) = w(\{h_2, h_3\}) = 0$. This means that $\{h_1, h_3\}$ violates h_2 but $\{h_3\}$ does not violate h_2, so locality fails. Defining $w(G) := v_G$ (as we have done it) 'saves' the situation. As an example that works with canonical w consider the smallest enclosing ball problem addressed above, in dimension 2. For a subset Q of the points we define $w(Q)$ as the radius of the smallest disk D_Q containing all the points in G. D_Q is unique, and G violates p if and only if p lies outside D_Q. From this, locality easily follows (and monotonicity is obvious). However, the situation is different for the smallest enclosing disk in the L_∞-norm (smallest enclosing axis-parallel square). Defining $w(Q)$ as the sidelength of the smallest square covering Q does not work, because this square is in general not unique and locality may fail, just like in LP. Computing the smallest enclosing axis-parallel rectangle instead (from which a smallest square can be obtained) fixes the problem.

These examples are not meant to suggest that similar transformations can be applied to create an LP-type problem out of any problem that just 'looks' LP-type. As an example, consider the *closest pair* problem. For a point set P in the plane and a subset Q, let $w(Q)$ be the closest interpoint distance in Q. Then w is monotone (the other way round) but locality fails (it is an easy exercise to verify this). Moreover, there is no way to cast the closest pair problem as an LP-type problem with fixed combinatorial dimension (if $w(Q)$ as above would work, we had combinatorial dimension 2), because then Theorem 8 would imply the existence of an $O(n)$ algorithm, $n = |P|$, contradicting the $\Omega(n \log n)$ lower bound.

5 A Subexponential Bound for LP-type Problems

Fix some LP-type problem (H, w) of combinatorial dimension δ, $|H| = n$. In the previous section we have argued that `SUBEX_lp` can solve the problem with an expected subexponential number of primitive operations, provided that calls to pairs (G, B) with $|G| = \delta$ are 'free'. This is the case if the problem is basis-regular, i.e. all bases have size exactly δ, like in LP with $\delta = d$. For many other problems (including the ones listed above), basis-regularity is not naturally satisfied. Here we prove that the subexponential analysis can in fact be extended to all LP-type problems, whether they are basis-regular or not. Let us start by 'pretending' that small problems are free; specifically, we assume existence of an algorithm `Small_LPtype` that can handle pairs (G, B) with $|G| \leq \delta$. Then any LP-type problem can be solved by the following algorithm `LPtype` which is just the abstract analogue of `SUBEX_lp`, with `Small_LPtype` plugged in as a subroutine.

```
function procedure LPtype(G, B)                          B ⊆ G: basis
    if |G| ≤ δ then                                       returns B_G
        return Small_LPtype(G, B)
    else
        choose random h ∈ G − B
        B' := LPtype(G − {h}, B)
        if h is violated by B' then
            return LPtype(G, basis(B', h))
        else
            return B'
```

If we evaluate the performance of `LPtype` in terms of violation tests, then the bounds developed for `SUBEX_lp` apply with a multiplicative overhead of $t_S(\delta)+1$, where $t_S(\delta)$ denotes the maximum expected number of violation tests incurred by algorithm `Small_LPtype` (which we develop later) on a single instance. This algorithm will – just like `LPtype` – have the property that any basis computation is preceded by a violation test, so bounding the latter bounds the former as well. Specifically, the following holds.

Lemma 9 *Given some initial basis $B \subseteq H$, a basis B_H of H can be computed with an expected number of no more than $(n-\delta)e^{O(\sqrt{\delta \ln n})}(t_S(\delta)+1)$ violation tests, resp. basis computations.*

Note that for basis-regular LP-type problems, $t_S(\delta)=0$. Below we prove that in the general case $t_S(\delta)$ can be bounded by

$$2\delta \exp\left(2\sqrt{\delta} + 2\sqrt[4]{\delta}\ln\delta + \ln^2\delta\right) = e^{O(\sqrt{\delta})}, \tag{3}$$

where the exponent is asymptotically smaller than the exponent we already get from the analysis of `SUBEX_lp`. Thus, the contribution of the small problems does not introduce asymptotic overhead in the exponent of Lemma 9. In particular, the best bound we were able to give for LP in Theorem 7 then applies to any LP-type problem.

Theorem 10 *Given an LP-type problem (H, w), $|H| = n$, of combinatorial dimension δ, and some initial basis $B \subseteq H$, a basis B_H of H can be computed with an expected number of no more than $O(\delta n + e^{O(\sqrt{\delta \ln \delta})})$ primitive operations.* ◻

For the concrete LP-type problems *smallest enclosing ball* and *polytope distance* mentioned above, this bound becomes a 'real' bound in terms of arithmetic operations, since in these cases a violation test can be performed in time $O(d)$ and a basis computation in time $O(d^3)$ [15].

The small problems. Recall that the crucial feature in the analysis of `SUBEX_lp` was that if $|G-B|$ is large, then the second recursive call is efficient on average due to a substantial decrease in hidden dimension. Our algorithm `Small_LPtype` exploits the same idea. It will have a first and a second recursive call, the first one solving the problem on $G-\{h\}$, for h randomly chosen from some sample space. However, if the canonical sample space $G-B$ turns out to be too small (which could be the typical situation for $|G| \le \delta$), some preliminary computations will enlarge it to make the second call more efficient on average. The degree of enlargement is obtained from a tradeoff between the extra effort and the gain in efficiency.

Fix some pair (G, B). If we would (like it is done in `LPtype`) insist on using the same basis B for the first recursive call on the set $G-\{h\}$, then of course, h would be restricted to lie in $G-B$, and there would be no way to enlarge the sample space. But we might as well recur with $(G-\{h\}, B')$, where $B' \subseteq G-\{h\}$ is some other basis. As long as B' is no worse than B (i.e. $w(B') \ge w(B)$ holds), such a choice can make the whole procedure only more efficient. In this situation, a call with $(G-\{h\}, B)$ *or* with $(G-\{h\}, B')$ can be performed for all $h \in G-B\cap B'$, which typically is a larger set than $G-B$. Thus, the more alternative bases B' are available, the larger the sample space gets; the 'preliminary computations' addressed above just represent a way of collecting enough such alternative bases. The following defines elements that might potentially end up in the sample space.

Definition 11 *$h \in B \subseteq G$ is called* free *with respect to (G, B) if there exists a basis $B' \subseteq G - \{h\}$ with $w(B') \geq w(B)$. In this case B' is called a* witness *for h with respect to (G, B).*

A witness of h proves that h is free. Elements of $G - B$ are free with witness B. With this terminology, our task consists of finding free elements (in addition to the ones in $G - B$, if their number does not suffice), along with corresponding witnesses.

To do this, we incrementally update a set D of free elements; initially, $D := G - B$. In case D is already large enough, which for our purposes will mean $|D| \geq \lceil \alpha |G| \rceil$, for $0 < \alpha < 1$ a constant to be determined later, there is nothing to do. Otherwise we will have to enlarge D by at least one more free element 'hidden' in B; to this end we step through a sequence of basis computations, until a witness for a yet unknown free element is discovered,[4] as follows: recursively call the algorithm with (G, B), but supply an additional parameter E that contains all the elements of G whose status is yet unknown (initially, $E = B$). This recursive call now has two ways to terminate: either it finds the desired basis B_G or – while improving its basis – discovers some $B' \subseteq G$ which fails to contain E. This, however, means that the elements in $E - B'$ have been uncovered as free elements with witness B', so the call has found at least one new free element and has therefore accomplished its task. Shrink E accordingly and repeat. The key observation is that as long as D is small, the set E of elements of unknown status will be large. Since the recursive call with parameter E terminates as soon as the first basis B' appears that is no longer wedged between E and G, it actually operates only on $G - E$ instead of G. This makes it substantially cheaper. In other words, the algorithm tentatively 'enforces' the elements of E and either finds the optimal basis in G under this tentative hypothesis, or it disproves the hypothesis by delivering (at least) one new free element. If enough free elements have been found (i.e. E has become small enough), the algorithm proceeds as before: it removes a random free element h and recurs on $(G - \{h\}, B^h)$, where B^h is a witness of h. This idea of tentatively enforcing constraints in order to find new free elements in an efficient manner has first been used by Kalai in one of his subexponential simplex algorithms for LP [17].

The Algorithm. The problem is solved by a call to an auxiliary algorithm `Small_LPtype_E` which has three parameters $E \subseteq B \subseteq G$.

```
function procedure Small_LPtype(G, B)          B ⊆ G: basis
    return Small_LPtype_E(G, B, ∅)             returns B_G
```

The top-level of `Small_LPtype_E` (where $E = \emptyset$) matches the description given above. Down the recursion the size of the problem is measured by $|G - E|$, and the quantity $\lceil \alpha |G - E| \rceil$ replaces $\lceil \alpha |G| \rceil$ as the critical value for the required number of free elements.

The procedure `Small_LPtype_E`(G, B, E) either returns B_G or delivers a basis B' with $w(B') > w(B)$ and $E \not\subseteq B' \subseteq G$ (if $E = \emptyset$, only the former alternative is possible, therefore `Small_LPtype`(G, B) indeed returns B_G). The current set D of free elements is implicitly maintained as $G - E'$. The witness for an element h will be denoted by B^h.

[4] or we already find an optimal basis B_G

```
function procedure Small_LPtype_E(G, B, E)          E ⊆ B ⊆ G; B: basis
    if G = B then                                returns B_G or basis B', with
        return B                                 w(B') > w(B), E ⊈ B' ⊆ G
    else
        E' := B
        for all h ∈ G − E' do B^h := B
        while |G − E'| < ⌈α|G − E|⌉ do
            B := Small_LPtype_E(G, B, E')
            if E ⊈ B then
                return B
            elsif E' ⊈ B then
                for all h ∈ E' − B do B^h := B
                E' := E' ∩ B
            else return B
        choose random h ∈ G − E'
        B' := Small_LPtype_E(G − {h}, B^h, E)
        if E ⊈ B' then
            return B'
        else
            if B' is violated by h then
                B'' := basis(B', h)
                if E ⊈ B'' then
                    return B''
                else return Small_LPtype_E(G, B'', E ∪ {h})
```

Termination of `Small_LPtype_E` follows by observing that the recursive calls solve smaller problems, measured in terms of $|G - E|$ (note that the first recursive call is not executed if $E' = E$ because then the algorithm does not enter the `while`-loop). The `while`-loop terminates because E' gets smaller in every iteration. The correctness of the algorithm follows by inductively checking the invariant '$E \subseteq B \subseteq G$'.

Let $t_E(m, k)$ denote the maximum expected number of violation tests during a call to algorithm `Small_LPtype_E`(G, B, E) of *size* $|G - E| = m$ and *freedom* at most k, where the freedom of a triple (G, B, E) is the number of free elements with respect to (G, B) that are *not* in E (this implies $k \leq m$). The following recurrence relation can be proved, invoking arguments similar to the ones used in the development of recurrence (2).

Lemma 12 *$t_E(m, 0) = 0$. For $m \geq k > 0$, $t_E(m, k)$ is bounded by*

$$\sum_{i=0}^{\lceil \alpha m \rceil - 1} t_E(i, \min(k, i)) + t_E(m-1, k-1) + 1 + \frac{1}{\lceil \alpha k \rceil} \sum_{i=1}^{\lceil \alpha k \rceil} t_E(m-1, k-i).$$

From this, an upper bound of

$$t_E(m, k) \leq 2 \exp\left(2\sqrt{k/\alpha} + \ln m (\log_{1/\alpha} m + 1)\right)$$

can be derived, and noting that $t_S(\delta) \leq t_E(\delta, \delta)$, we obtain (with $\alpha := 1/(1 + \ln \delta / \sqrt[4]{\delta})$)

$$t_S(\delta) \leq 2\delta \exp\left(2\sqrt{\delta} + 2\sqrt[4]{\delta} \ln \delta + \ln^2 \delta\right),$$

which is the bound claimed in (3) for the maximum expected number of violation tests incurred by a call to `Small_LPtype`(G, B). For the details of the analysis see [14, 15].

Abstract optimization problems. Let us investigate the properties crucially underlying algorithm `Small_LPtype`. As it turns out, these are surprisingly few. First, the basis B'' need not be a basis of $B' \cup \{h\}$. It suffices that $w(B'') > w(B')$. In case of LP this is equivalent to $B'' = B_{B' \cup \{h\}}$, but in a general LP-type problem, many bases B'' with the required property might exist. In this situation, any of them would do. Thus, the basis computation primitive could as well be replaced with a *basis improvement* primitive which might be (much) easier to implement. Anticipating this relaxation, the $O(d^3)$ bounds stated above for the basis computation in the *smallest enclosing ball* and *polytope distance* problems were actually bounds for the basis improvement [15].

Second, in order to guarantee the invariant '$E \subseteq B \subseteq G$', the improved basis B'' has to be a subset of G but not necessarily of $B' \cup \{h\}$. By the locality axiom we know that $B' \cup \{h\}$ must contain an improved basis if G contains one (and the correctness of the algorithm in its present form crucially relies on this fact), but if we had a method of directly accessing some arbitrary improved basis in G, the algorithm would work completely without the locality assumption.

Third and last, the function w is quite irrelevant. The important feature is that the bases are linearly ordered, and solving the problem for a set $G \subseteq H$ means finding the largest basis among all bases which are subsets of G. We can as well assume that any subset of H actually is a basis.

Following this discussion, we define an *abstract optimization problem* (AOP) to be a triple

$$(H, \preceq, \Phi),$$

where H is a finite set, $\preceq$ is some linear ordering of 2^H and Φ is an *improvement oracle* which is a function on pairs $(G, B), B \subseteq G \subseteq H$ with the following property.

$$\Phi(G,B) = \begin{cases} B, \text{ if } B = \max_{\preceq}\{B' \mid B' \subseteq G\}, \\ B' \succ B, B' \subseteq G, \text{otherwise.} \end{cases}$$

Under this model, the optimal (with respect to $\preceq$) subset B_H of H can be found with at most $2^n - 1$ calls to the improvement oracle, where $n = |H|$. However, we can almost verbatim apply algorithm `Small_LPtype` to solve an abstract optimization problem in subexponential time. To this end, violation test and basis computation are replaced by a single call to the oracle: compute $B'' := \Phi(G, B')$. Then the violation test becomes a query '$B'' \neq B'$?', and if this query has a positive answer, the improved basis B'' is readily available. The careful reader might have noticed that under the AOP axioms, $G = B$ does not necessarily imply that B is optimal for G, so the termination criterion checked in the first line of the algorithm is not valid. However, we can assume the following stronger oracle.

$$\Psi(G,B) = \begin{cases} B, \text{ if } B = \max_{\preceq}\{B' \mid B' \subseteq G\}, \\ B' \succ B, B' \subseteq G, B' = \max_{\preceq}\{B'' \mid B'' \subseteq B'\}, \text{otherwise,} \end{cases}$$

which guarantees that only bases are considered during the algorithm which define their own optimum. $\Psi(G, B)$ can be simulated with at most $|G|$ calls to Φ (replace the basis B' resulting from $\Phi(G, B)$ with $\Phi(B', B')$ until B' is stable). Invoking the bound (3) for `Small_LPtype` we thus get

Theorem 13 *Given an AOP $(H, \preceq, \Phi)$, a set $B_H = \max_{\preceq}\{B \mid B \subseteq H\}$ can be computed with an expected number of no more than*

$$2n^2 \exp\left(2\sqrt{n} + 2\sqrt[4]{n} \ln n + \ln^2 n\right)$$

calls to the improvement oracle Φ.

If not before, the power of randomization in our context becomes apparent here. Namely, the following lower bound can be shown, see [14, 15].

Theorem 14 *Let $\mathcal{A}$ be a deterministic algorithm for solving AOPs on an n-element set H. Then there exists an AOP $(H, \preceq, \Phi)$ which forces $\mathcal{A}$ to call Φ at least $2^n - 1$ times to find the optimal subset B_H.*

For LP-type problems, no similar (and even no superpolynomial) lower bound for deterministic algorithms is known.

6 Discussion

We have reviewed several (randomized) algorithms for LP and LP-type problems which in combination lead to the currently best known subexponential bounds.

A challenging open problem is to improve on these bounds, and to prove or disprove the existence of a polynomial combinatorial algorithm for linear programming. While algorithm `SUBEX_lp` might still substantially outperform the bound of Lemma 6 on actual LPs, Matoušek has constructed abstract LP-type problems which force `SUBEX_lp` to behave not much better than analyzed [21]. An interesting problem in this context is to determine the complexity of AOPs. Due to the very weak and simple axioms it might be possible to prove a nontrivial (superpolynomial) lower bound on the number of improvement oracle calls needed by any algorithm for solving AOPs. For deterministic algorithms, such a bound is given by Theorem 14, for randomized algorithms not even a superlinear bound (w.r.t. $|H|$) is known. On the other hand, any improved upper bounds for AOPs immediately apply to small instances of LP ($n = O(d)$) and the other concrete LP-type problems we have mentioned.

Chazelle and Matoušek [5], gave a deterministic algorithm solving LP-type problems in time $O(\delta^{O(\delta)}n)$, provided an additional axiom holds (together with an additional computational assumption). Still these extra requirements are satisfied in many LP-type problems.

A natural generalization is the problem of finding the best solution satisfying all but k of the given constraints, for abstract LP-type problems as defined here. This has been investigated by Matoušek [22].

Amenta [2], considers the following extension of the abstract framework: Suppose we are given a family of LP-type problems (H, w_λ), parameterized by a real parameter λ; the underlying ordered value set $\mathcal{W}$ has a maximum element ∞ representing *infeasibility*. The goal is to find the smallest λ for which (H, w_λ) is feasible, i.e. $w_\lambda(H) < \infty$. [2] provides conditions under which the such a problem can be transformed into a single LP-type problem, and she gives bounds on the resulting combinatorial dimension. Related work can be found in [3].

References

[1] Adler, I. and Shamir, R.: A randomized scheme for speeding up algorithms for linear and convex programming problems with high constraints-to-variables ratio. Math. Programming **61** (1993) 39–52

[2] Amenta, N.: Helly-type theorems and generalized linear programming. Discrete Comput. Geom. **12** (1994) 241–261

[3] Amenta, N.: Bounded boxes, Hausdorff distance, and a new proof of an interesting Helly-type theorem. Proc. 10th Annu. ACM Symp. Computational Geometry (1994) 340–347

[4] Borgwardt, K. H.: The Simplex Method. A Probabilistic Analysis. Volume 1 of *Algorithms and Combinatorics*, Springer-Verlag, Berlin-Heidelberg (1987)

[5] Chazelle, B., Matoušek, J.: On linear-time deterministic algorithms for optimization problems in fixed dimensions. Proc. 4th SIAM-ACM Symp. on Discrete Alg. (1993) 281–290

[6] Chvátal, V.: Linear Programming. W. H. Freeman, New York, NY (1983).

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

[8] Clarkson,K. L.: A Las Vegas algorithm for linear and integer programming when the dimension is small. J. ACM **42**(2) (1995) 488–499

[9] Dantzig, G. B.: Linear Programming and Extensions. Princeton University Press, Princeton, NJ (1963).

[10] Danzer, Über ein Problem aus der kombinatorischen Geometrie. Arch. Math. **8** (1957) 347–351

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

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

[13] Dyer, M. E., Frieze, A. M., A randomized algorithm for fixed-dimensional linear programming. Math. Programming **44** (1989) 203–212

[14] Gärtner, B.: A subexponential algorithm for abstract optimization problems. SIAM J. Comput. **24** (1995) 1018–1035

[15] Gärtner, B.: Randomized Optimization by Simplex-Type Methods. PhD thesis, Institute for Computer Science, Free University Berlin (1995)

[16] Goldwasser, M.: A survey of linear programming in randomized subexponential time. ACM-SIGACT News **26**(2) (1995) 96–104

[17] Kalai, G.: A subexponential randomized simplex algorithm. Proc. 24th Annu. ACM Symp. Theory of Computing (1992) 475–482

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

[19] Klee, V., Minty, G. J.: How good is the simplex algorithm? In O. Shisha, editor, *Inequalities III*, Academic Press (1972) 159–175

[20] Matoušek, J., Sharir, M., Welzl, E.: A subexponential bound for linear programming. Proc. 8th Annu. ACM Symp. Computational Geometry (1992) 1–8; Algorithmica, to appear

[21] Matoušek, J.: Lower bounds for a subexponential optimization algorithm. Random Structures & Algorithms **5**(4) (1994) 591–607

[22] Matoušek, J.: On geometric optimization with few violated constraints. Proc. 10th Annu. ACM Symp. Computational Geometry (1994) 312–321

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

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

[25] Shrijver, A.: Theory of Linear and Integer Programming. Wiley, New York (1986)

[26] Sharir, M., Welzl, E.: A combinatorial bound for linear programming and related problems. Proc. 9th Symp. Theo. Asp. Comp. Sci., Lecture Notes in Computer Science **577** (1992) 569–579

[27] Sharir, M., Welzl, E.: Rectilinear and polygonal p-piercing and p-center problems. Manuscript, submitted (1995)

[28] Sharir, M., Welzl, E.: Circular and spherical separability. Manuscript, submitted (1995)