A Subexponential Randomized Simplex Algorithm

(Extended Abstract)

Gil Kalai *
Institute of Mathematics, Hebrew University of Jerusalem, Jerusalem, Israel

Abstract

We describe a randomized variant of the simplex algorithm. The expected number of arithmetic operations required by our algorithm for every linear programming problem with d variables and n constraints is at most

$$min\{n^{C\cdot\sqrt{d/\log d}}, n2^{C\sqrt{n-d}\sqrt{\log d}}\}.$$

(C is an absolute constant.)

1 Introduction

In 1950 George Dantzig [7, 8] introduced the *simplex algorithm* for solving linear programming problems. Linear programming and the simplex algorithm are among the most celebrated applications of mathematics. The reader is referred to [28] for a very thorough description of linear programming theory including background material, historical comments, and description of recent developments. For a quick introduction to linear programming and the simplex algorithm see [4].

Linear programming (briefly, LP) is the problem of maximizing a linear function (called the *objective function*) of d variables, subject to n linear constraints. The set of points which satisfy the constraints form a convex polyhedron called the *feasible polyhedron*. If it exists, the maximum of the objective function is attained at a vertex of the feasible polyhedron (briefly, a *feasible vertex*). Given the feasible polyhedron P, the *graph* of P, denoted by $G(P)$, is an abstract graph whose vertices are the vertices of P, where two vertices are adjacent if they form the end points of a 1-dimensional face of P.

Each step of the simplex algorithm consists of moving from one feasible vertex v to an adjacent feasible vertex w which has a higher value of the objective function. The precise choice of w, given v, is called the *pivot rule*. The simplex algorithm is, in fact, a name for a whole class of algorithms depending on the specific pivot rules. Many pivot rules were suggested and tested over the years.

The performance of the simplex algorithm is extremely good in practice. In the early days of linear programming it was believed that the common pivot rules reach the maximal vertex in a number of steps which is polynomial in d and n. See e.g., [8, 11]. It came as a great surprise when Klee and Minty [21] found in 1971 that one of the most common variants of the simplex algorithm is exponential in the worst case. In fact, the number of steps was quite close to the total number of vertices of the feasible polyhedron. Similar results for other pivot rules were subsequently found by several authors, see [20] for references.

Dantzig was also the first to consider randomized pivot rules. He (and others) suggested to choose in each step the next vertex at random among the neighboring vertices which improve the value of the objective function. The hope was that a randomized pivot rule of this kind will require for every linear programming problem an expected number of steps which is polynomial (or at least subexponential).

Until now no pivot rule (either deterministic or randomized) for the simplex algorithm with subexponential worst case behavior was known. In this paper we describe a randomized pivot rule (different from the

*Supported in part by a BSF grant, by a GIF grant and by the Mittag-Leffler Institute. The work described here was done, in part, while the author was a visiting scientist at IBM Almaden Research Center in San Jose.

24th ANNUAL ACM STOC - 5/92/VICTORIA, B.C., CANADA
© 1992 ACM 0-89791-512-7/92/0004/0475...$1.50

rule described above) which takes on every linear program with d variables and n constraints an expected subexponential number of arithmetic operations.

Let $\mathcal{L}(d,n)$ denotes the class of linear programming problems with d variables and n constraints. For $O \in \mathcal{L}(d,n)$ let $f(O)$ be the expected number of pivot steps performed by the algorithm for the problem O and let $f(d,n)$ be the maximum of $f(O)$ over all $O \in \mathcal{L}(d,n)$. The number of arithmetic operations per pivot step will be at most $d^2 \cdot n \log n$.

The following result describes the behavior of $f(d,n)$.

Theorem 1

$$f(d,n) \leq n^{O(\sqrt{d/\log d})}, \tag{1}$$

$$f(d,2d) \leq 1000^{\sqrt{d}}, \tag{2}$$

$$f(d,d+m) \leq 2^{O(\sqrt{m \log d})}. \tag{3}$$

Here is a quick rough description of the algorithm. Let O be a linear programming problem with d variables and n constraints and let v be a feasible vertex.

Stage I: Starting from v, reach vertices in at least r facets. Stage II: Choose a facet F at random among these r facets and apply the algorithm recursively to find an optimal vertex w on F. Then, set $v =: w$ and repeat (from Stage I).

To get best results we choose $r = max(d, n/2)$ (and this value is updated throughout the algorithm.) The algorithm obtained by choosing $r = d$ satisfies relation (3) but not relations (1,2). For this choice the first stage is automatic and the algorithm becomes:

Choose a facet F containing v at random, apply the algorithm recursively to reach an optimal vertex w in F. Set $v =: w$ and repeat.

The basic algorithm and several variants are described in Section 3.

The starting point of our research was the diameter problem for graphs of polyhedra. See [14, 15, 16]. This is discussed in Section 2, where it is proved that if we do not restrict the complexity needed to find the pivot steps, $n^{\log d}$ pivot steps always suffice.

Let us put Theorem 1 in the wider context of the complexity of linear programming. In a major breakthrough Khachiyan [18] found in 1979 a polynomial algorithm for linear programming. Khachiyan showed that the ellipsoid algorithm (which was proposed by Yudin, Nemirovskii and Shor - for nonlinear optimization,) requires a polynomial $p = p(d,n,L)$ number of arithmetic operations on every LP problem in $\mathcal{L}(d,n)$. Here, L denotes the total input size of the problem. Better polynomial bounds were found by Karmarkar [17] and others. Karmarkar also discovered that variants of his algorithm are extremely quick in practice. (This started an era of competition between different LP algorithms, an era marked by the discovery of many practically quick interior point algorithms for linear programming as well as substantial improvements in the performance of variants of the simplex algorithm itself.)

In the early days of linear programming people were interested in the number of arithmetic operations as a function of d and n. The dependence of Katchiyan's bound also on L came with some surprise. The problem of finding the complexity of linear programming in terms of the number of arithmetic operations needed as a function of d and n alone is still wide open. Theorem 1 implies, for the first time, that the complexity of linear programming in these terms (using randomization) is subexponential.

Call an algorithm (deterministic or randomized) for linear programming *strong* if for every problem in $\mathcal{L}(d,n)$ the number of arithmetic operations is bounded by a function of d and n. (I.e., it is independent from the number of bits needed to describe the constraints and the objective function.) It is an outstanding open problem to find a strong polynomial algorithm for linear programming. (Or to prove that no such algorithm exists.) See [26]. Important progress in finding strong polynomial algorithms for linear programming problems of special types were achieved by Megiddo [26], Tardos [31] and others.

Theorem 1 shows the existence of a strong subexponential algorithm for linear programming. The existence of a strong polynomial (or quasi-polynomial) algorithm for linear programming and the existence of a strong subexponential algorithm for linear programming which is deterministic remain open.

Megiddo [27] found the first algorithm for linear programming which is linear for a fixed dimension. The complexity of his algorithm is $2^{2^d}n$. (Improved algorithms which requires $3^{d^2}n$ arithmetic operations were found by Dyer [9] and by Clarkson [5].) Clarkson [6] found a randomized algorithm which takes an expected $O(d^{\log\log n} \cdot d^d) + O(d^2 n)$ arithmetic operations. Seidel [29] found a very simple randomized algorithm which takes $O(d!n)$ operations.

Clarkson's algorithm (with tiny changes) is a dual-simplex algorithm. It uses the simplex algorithm with d variables and d^2 constraints as a subroutine. By plugging our algorithm into Clarkson's algorithm we get a randomized dual-simplex algorithm which requires an expected

$$d^{O(\sqrt{d/\log d}+\log\log n)} + O(d^2 n) \tag{4}$$

arithmetic operations.

2 The shortest path to the top

Let P be the feasible polyhedron of a linear programming problem O in $\mathcal{L}(d,n)$. Let ϕ denote the linear objective function. Let $G(P)$ denote the graph of P and let $\vec{G}(O)$ denote the directed graph obtained from $G(P)$ by directing an edge $\{v,w\}$ from v to w if $\phi(v) < \phi(w)$. If ϕ is not bounded on P we add a vertex ∞ (at infinity) to $\vec{G}(O)$ such that there is a directed edge from a vertex v to ∞ if there is a 1-dimensional face containing v on which ϕ is not bounded from above.

Let $\delta(G(P))$ be the diameter of the graph $G(P)$.

For a vertex v of $\vec{G}(O)$ let $h(v)$ denote the minimal length of a directed path from v to a vertex w for which ϕ attains its maximum, or to the vertex ∞ if ϕ is unbounded. The *height* of $\vec{G}(O)$ denoted by $\delta(\vec{G}(O))$ is the maximum of $h(v)$ over all vertices v of $\vec{G}(O)$.

Let $\Delta(d,n)$ and $H(d,n)$ denote respectively the maximum over $\delta(G(P))$ and $\delta(\vec{G}(O))$ over all O in $\mathcal{L}(d,n)$. It is easy to see that $\Delta(d,n) \leq H(d,n)$. $H(d,n)$ is the minimal number of pivot steps needed for a simplex algorithm in which we allow an unlimited computation power for producing the pivot steps.

Hirsch conjectured, see [8, 20], that $\Delta(d,n) \leq n-d$. Klee and Walkup [22] gave examples of (unbounded) polyhedra which violate the Hirsch conjecture. Their examples give $\Delta(d,n) \geq n-d+\lfloor d/5 \rfloor$ and this is the best known lower bound for $\Delta(d,n)$. The Hirsch conjecture is still open for bounded polyhedra.

Larman [23] proved that $\Delta(d,n) \leq n \cdot 2^{d-3}$. Until recently, no subexponential upper bound for $\Delta(d,n)$ was known. In [15] the author proved a quasi polynomial bound $\Delta(d,n) \leq n^{2\log d+3}$. Kleitman, see [16], found a substantial simplification which also slightly improved the upper bound to $n^{\log d+1}$. We describe here a further modification of Kleitman's argument which applies to $H(d,n)$ as well.

For a vertex $w \in \vec{G}(O)$ let $U(w)$ denote the set of facets F of P such that $max\{\phi(x) : x \in F\} > \phi(w)$. We call the facets on $U(w)$ the *active* facets of P with respect to w. Let $u(w) = |U(w)|$. Let $\bar{H}(d,n)$ denote the maximum value of $h(v)$ over vertices v of $\vec{G}(O)$ which satisfy $u(v) \leq n$ where O ranges over all linear programming problems with d variables (with arbitrary number of constraints). Clearly $H(d,n) \leq \bar{H}(d,n)$.

Theorem 2

$$\bar{H}(d,n) \leq n \cdot \binom{d+\log n}{\log n}. \qquad (5)$$

Proof: Let O be an arbitrary linear programming problem with d variables. Let v be a vertex of the feasible polyhedron P with $u(v) \leq n$. Consider a subset $S \subset U(v)$ of size k. We would like to give an upper bound B for the length of the shortest monotone path from v which either reaches the top of P or reaches a vertex which belongs to some of the facets in S. Clearly if v is a vertex of some of the facets in S, then $B = 0$ will do. Assume that v is not a vertex of any $F \in S$ and consider the linear programming problem O' obtained by ignoring all inequalities corresponding to facets in S, and to all non-active facets which do not include the vertex v. Let P' be the feasible polyhedron of O'. Note that v is a vertex of P', nonactive facets of P which contain the vertex v correspond to non-active facets of P', and therefore $u(v) \leq n-k$. In P' there is a monotone path from v to a top edge of length at most $\bar{H}(d,n-k)$. If this path is in P we get $h(v) \leq \bar{H}(d,n-k)$. If not consider the first edge $\{x,y\}$ in the path so that x is a vertex of P and y is outside P. The last point of P in this edge is a vertex of P which belongs to one of the facets of S. So $B \leq \bar{H}(d,n-k)$.

We have seen that for every family S of k facets we can reach a vertex in one of the facets with a monotone path of length $\bar{H}(d,n-k)$. It follows that with a monotone path of length $\bar{H}(d,n-k)$ we can reach vertices in at least $n-k+1$ facets. With a monotone path of length $\bar{H}(d,n-k)+\bar{H}(d-1,n-1)$ we can reach a vertex with a maximal value of ϕ in those $n-k+1$ facets. A vertex W with a maximal value of ϕ among all such vertices will satisfy $u(w) \leq k-1$. From this vertex to reach the top requires $H(d,k-1)$. Put $k = \lfloor d/2 \rfloor$ to obtain

$$H(d,n) \leq H(d-1,n-1) + 2H(d,\lfloor n/2 \rfloor). \qquad (6)$$

Define $f(d,t) = 2^t H(d,2^t)$. Relation (6) gives that $f(d,t) \leq f(d-1,t)+f(d,t-1)$ and therefore $f(d,t) \leq \binom{d+t}{d}$. Therefore $H(d,n) \leq n \cdot \binom{logn+d}{logn}$.

Remarks: 1. Since $\Delta(d,n)$ is linear in n it follows that, for a fixed dimension, every (primal) simplex algorithm needs at least a linear number in n of pivot steps. The upper bound (5) is slightly superlinear in n for a fixed d. It would be interesting to find an upper bound on $H(d,n)$ which is linear in n for a fixed d.

If $m = n-d$ is fixed and $n \leq 2d$ then it is immediate that $H(d,n) \leq H(m,2m)$. Therefore, for a fixed d and large n there may be a dual simplex algorithm for which the number of pivot steps is completely independent from n. Clarkson's algorithm is (essentially) a dual-simplex algorithm where the dependence of the number of pivot steps on n is highly sublinear. Klee and Minty asked [21] if *every* monotone path in $\vec{G}(O)$ is of length at most $C(m) \cdot n$. (Or, in other words, is it true that, for a fixed dimension, *every* dual-simplex pivot rule takes a linear number of pivot steps.)

2. The recurrence (6) implies slightly better bounds when n is close to d, e.g., $\bar{H}(d, 2d) \leq d^{2+\log d/\log_2 9}$.

3. The monotone path of length l guaranteed by Theorem 2 can be found by an algorithm which calls $n^2 \cdot l$ times to a linear programming subroutine.

4. The proof of Theorem 2 and the algorithm described in the next section apply to the abstract combinatorial setting of a shellable simplicial complex with a fixed shelling order. Compare [15], Sec. 4.

3 How to get to the top - the algorithm

We will assume that we have a linear programming problem $O \in \mathcal{L}(d,n)$ with a vertex of the feasible polyhedron and our algorithm will give the top vertex of the feasible polyhedron. In the description of the algorithm we assume that the feasible polyhedron is simple, i.e., every vertex is included in exactly d facets. (In other words, the problem is nondegenerate.) This is no loss of generality and, in fact, slight changes in the description of the algorithms (e.g. to consider feasible bases rather than feasible vertices) are needed for the general case.

Given a vertex v a step in the algorithm is to choose an improving edge of the polyhedron for which v is an end point and to move to the other end point. Typically, we will deal with a different linear program O' obtained from the original problem by ignoring some of the original inequalities. However in each pivot step we will move only until reaching a vertex of the original polyhedron. The complexity of each step is bounded by d^2n. (There are also at most one per pivot choices of a random number between 1 and n.)

Remark: We will describe three variants of the algorithm - $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_0$ in the chronological order in which they were analysed. Algorithm $\mathcal{S}_1$ has a clumsy description but given its description it is almost evident that it is subexponential. $\mathcal{S}_2$ gives the best expected running time, and $\mathcal{S}_0$ has the simplest description. (The value of $f(d,n)$ in Theorem 1 is the complexity of algorithm $\mathcal{S}_2$.)

We will describe now algorithm $\mathcal{S}_1$ and denote by $f(d,n)$ the expected number of pivot steps needed for every problem O in $\mathcal{L}(d,n)$. The algorithm is carried on in three stages:

Stage I: *finding vertices on many facets.* Suppose you know vertices on k facets. Ignore all inequalities which correspond to all the other facets, and solve recursively the remaining LP problem O'. If the optimal point is feasible it is optimal for O and the algorithm terminates. Otherwise, consider the first edge of the walk which leave the feasible polyhedron (for the original problem). The last feasible point on that edge will be a feasible vertex which belongs to a new facet.

So, after $\sum_{i=d}^{r} f(d,i)$ operations you find vertices in r facets.

Stage II: *finding the optimum on a random facet.* Choose a facet at random from the r facets in which vertices were found. Then apply the algorithm on this facet, until reaching its top vertex w. This takes $f(d-1, n-1)$ steps.

Stage III: Let l be the number of active facets w.r.t. w, i.e., facets whose top vertex is above w. Try as in stage I to find vertices above w in r different facets. After $\sum_{i=d}^{r} f(d,i)$ steps you either found vertices in r facets or reached the optimal vertex of P. If $r > l$ you must have reached the top vertex of P and the algorithm terminates.

The algorithm as described above is not exactly a simplex algorithm since at certain points we have to backtrack to vertices which were reached in previous steps. It can easily be modified to an honest simplex algorithm. For this you choose the random facet F among the n facets in advance and start Stage II as soon as you reach a vertex on that facet. If you reached more than r facets non of which is F just start again stage I from the vertex you reached latest.

In the worst case, above the optimal vertex reached at Stage II there are with probability $1/r$, vertices in $n-i-1$ facets, $i = 1, 2, ..., r$. Therefore the expected total number of steps performed is at most

$$2\sum_{i=d}^{r} f(d,i) + f(d-1,n-1) + \frac{1}{r} \sum_{l=n-r-1}^{n-1} \sum_{i=d}^{l} f(d,n-i) \tag{7}$$

We modify now Stage III as follows. If it turns out that l is larger then $(1-c)n$, ($c > 1/2$ will be determined later and will be rather small), start the algorithm again from the vertex you have reached the latest. We also set $r = max\{n/2, d\}$. The resulting algorithm is $\mathcal{S}_1$ and its analysis is very simple.

The expected number of steps in the algorithm $\mathcal{S}_1$ is at most

$$2(1-c)^{-1} \sum_{i=d}^{r} f(d,i) + (1-c)^{-1} f(d-1,n-1) \tag{8}$$

$$+(1-c)^{-1}n^{-1}\sum_{i=[cn]}^{r} f(d,n-i)i$$

$$\leq 6nf(d,(1-c)n)+(1-c)^{-1}f(d-1,n-1).$$

In other words,

$$f(d,n)\leq(1-c)^{-1}f(d-1,n)+6nf(d,(1-c)n). \quad (9)$$

Now define $g(d,n)=(1-c)^{d}\cdot(6n)^{log_{1-c}n}f(d,n)$. We have $g(d,n)\leq g(d-1,n)+g(d,(1-c)n)$. Define further, $h(d,y)=g(d,(1-c)^{-y})$. $h(d,y)\leq h(d-1,y)+h(d,y-1)$. Therefore $h(d,y)\leq\binom{d+y}{y}$. Writing $b=1/(1-c)$ we get

$$f(d,n)\leq b^{d}(6n)^{log_b n}\binom{d+log_b n}{log_b n}. \quad (10)$$

For $c=1-1/\sqrt{d}$ we obtain

$$f(d,n)\leq 2^{\sqrt{d}}\cdot(6n)^{\sqrt{d}\log n}d^{\sqrt{d}\log n}.$$

If we let $c=c(d,n)=max\{1/2,1-\log n/\sqrt{d}\}$ we get from relation (9) (by a slightly more involved computation) that

$$f(d,n)\leq n^{16\sqrt{d}}. \quad (11)$$

Remark: The reader may wonder what is the reason for the different asymptotic behavior between the shortest monotone path, and the expected number of steps in the algorithm. The reason seems to be that if a sequence $a_1,a_2,a_3,\cdots,a_n,\cdots$ is defined by the rule: $a_{n+1}=a_n+a_{n/2}$, then a_n is roughly $n^{C\cdot\log n}$. But if the rule is changed to $a_{n+1}=a_n+(a_1+a_2+\cdots+a_n)/n$ then a_n is roughly $e^{2\cdot\sqrt{n}}$. (For an inductive proof of this fact estimate sums of the form $\sum e^{\sqrt{i}}$ by an integral of the function $e^{\sqrt{x}}$. This is the way to handle also relations (15) and (16) below.)

We will describe now several variants of Algorithm $\mathcal{S}_1$. Stage III of the algorithm can be deleted. After performing Stages I and II we can start again from the vertex we reached. After stage I and II are performed we are left with $n-i$ active constraints, where, i is, with equal probability, a number between 1 and r. One difficulty is that we do not know the exact number of constraints which are active at this stage. But we can use the expected value which is $n-r/2$. Another difficulty is that we do not know the exact $n-i$ constraints which are involved. But again this is not a serious problem since we can perform the algorithm as it is recursively without knowing the exact identity of the active constraints. Let $\mathcal{S}_2$ be the resulting algorithm, and let $f_2(d,n)$ be the expected number of pivot steps taken by it. The analysis of $\mathcal{S}_2$ (which is somewhat more involved) give slightly better estimates summerized by:

$$f_2(d,Kd)\leq 2^{O(\sqrt{d}\sqrt{K})}, \quad (12)$$

$$f_2(d,n)\leq n^{O(\sqrt{d}/\sqrt{\log d})}. \quad (13)$$

$$f_2(d,d+m)\leq 2^{O(\sqrt{m\log d})} \quad (14)$$

The last relation describes the behavior of the algorithm when the codimension is small, or, in other words, as a dual-simplex algorithm for problems with m variables and $m+d$ constraints.

These estimates are proved by studying the behavior of the recurrence relation

$$g(d,n)\leq g(d-1,n-1)+ \quad (15)$$

$$\sum_{i=d}^{n/2}g(d,i)+2/n\sum_{i=1}^{n/2}g(d,n-i).$$

We set now the parameter r in algorithm $\mathcal{S}_2$ to be d rather than $max\{d,n/2\}$. Stage I is deleted and the resulting algorithm $\mathcal{S}_0$ has the following simple description:

Starting from a vertex v of the feasible polyhedra, choose a facet F containing v at random, find the top vertex w in F (using the algorithm recursively), set $v=:w$ and repeat.

(A more general class of randomized algorithms was considered, but not analysed, by the auther already in [15], see Remark 6.2.) Let $f_0(d,d+m)$ be the expected number of pivot steps taken by $\mathcal{S}_0$ for a problem with d variables and $n=d+m$ constraints. One gets

$$f_0(d,d+m)\leq \quad (16)$$

$$\leq f_0(d-1,d+m-1)+1/d\sum_{i=0}^{d}f_0(d,d+m-i).$$

This implies (taking into account that that $f_0(d,n)=0$ for $n<d$)

$$f_0(d,d+m)\leq\exp(O(\sqrt{m}\cdot\log d)). \quad (17)$$

This bound is good whenever m is not too large with respect to d, and when m is small with respect to d it is similar to the bound for algorithm $\mathcal{S}_2$. (When n is large w.r.t. d we get a rather poor bound, which becomes meaningless when $n=d^2$.) If m and d are similar (say, $n=2d$) then algorithm $\mathcal{S}_2$ takes $\exp(O(\sqrt{d}))$ steps, rather than $\exp(O(\sqrt{d}\log d))$ taken by $\mathcal{S}_0$. Another way to save the $\sqrt{\log d}$ is to use algorithm $\mathcal{S}_0$ and to alternate between the primal and dual problems to guarantee that at each given stage the number of constraints is at most twice the number of variables. Recurrence (16) with the additional relation

$f_0(d, d+m) = f_0(m, d+m)$ suffices to imply that $f_0(d, 2d) \leq \exp O(\sqrt{d})$.

Remark: Note that with an oracle which gives the best possible choices for the random choices, algorithms $\mathcal{S}_1$ and $\mathcal{S}_2$, become quasi-polynomial and thus demonstrate the result of Section 2. We do not know this for $\mathcal{S}_0$. (Algorithm $\mathcal{S}_0$ with the alternations between primal and dual problems becomes linear under such an oracle.)

Plugging our algorithm into Clarkson's algorithm [6] for linear programming, we obtain a dual simplex algorithm which requires for a linear programming problem with d variables and n constraints,

$$(d^{O(\sqrt{d/\log d}+loglogn)}) + O(d^2 n), \tag{18}$$

arithmetic operations. A careful analysis of our algorithm for the case that $m = n - d$ is fixed, shows that the algorithm is linear in n for a fixed m but not as good as (18).

Remark: If $d = 1{,}000{,}000$ and $n = 2d$ then the feasible polyhedron may have more than $10^{400,000}$ vertices. Our algorithm requires at most an expected $10^{3,000}$ pivot steps. The quickest path to the top is of length less than 10^{50}.

4 The different paths to a subexponential algorithm

In independent work [30] Micha Sharir and Emo Welzl found a remarkable combinatorial randomized algorithm for Linear Programming (and more general problems). Their algorithm is dual-simplex and seems related to a dual form of the variant $\mathcal{S}_0$ of our algorithm. The Sharir-Welzl algorithm was motivated by several algorithms in computational geometry and, in particular, by Seidel's algorithm [29] for Linear Programming. (But not at all by the diameter problem for polyhedra.) Sharir and Welzl showed that the (expected) complexity of their algorithm is $O(n \cdot 2^d)$.

Later, following the results of our paper, Matoušek, Sharir and Welzl [24] were able to improve the analysis of the Sharir-Welzl algorithm and to get subexponential bounds which are similar to those given by equation (17). It is remarkable to see how different paths have led to rather similar results so close in time.

5 Open Problems

Here is a list of open problems. Some of the problems are discussed in more details in other parts of the paper.

1. Decide if $\Delta(d, n)$ and $H(d, n)$ are polynomial.
2. Get better randomized pivot rules. Can you push down the expected running time to $\exp(n^{1/3})$?? to $\exp(\log^2 n)$???
3. Find a deterministic subexponential pivot rule.
4. (Perhaps easy) Decide if $H(d, n)$ is linear in n for a fixed dimension d.
5. (Looks doable) Give a randomized (primal) pivot rule for which for a fixed dimension the complexity is at most $O(n^2)$, or $O(n^{200})$.
6. Is random pivoting subexponential?
7. For $O \in \mathcal{L}(d, n)$ let $\gamma(O)$ be the *maximal* length of a monotone path in $\vec{G}(O)$. a. [21] For a fixed $m = n - d$ is $\gamma(O) \leq O(n)$? b. For O chosen at random, is $\gamma(O)$ polynomial in d and n?

6 Concluding Remarks

1. *Random pivoting and random walks.* We describe and discuss here several randomized pivot rules. A simple randomized pivot rule $\mathcal{A}$ called ***random pivoting*** (suggested by Dantzig and others) is: **Choose a vertex w at random among the neighboring vertices of v which improve the value of the objective function. Set $v =: w$ and repeat.** Algorithm $\mathcal{A}$ can be considered for every directed acyclic graph. It is an interesting problem (suggested by several people) to give conditions for random walks on directed acyclic graphs to reach a sink "quickly".

Related rules based on random walks were suggested in [10, 14]. In [10] Dyer and Friese use such a pivot rule to give a randomized polynomial algorithm for an important class of linear programming problems. Given a vertex v of the feasible polyhedra P, let $G[v]$ be the graph induced from $G(P)$ (the graph of P) on all vertices of P with higher or equal value of the linear objective function. The algorithm $\mathcal{B}[t]$ from [14] is described by: **Start from the vertex v a random walk of length t on the graph $G[v]$ reaching a vertex w. Set $v =: w$ and repeat.** Thus, $\mathcal{B}[1] = \mathcal{A}$. The idea is to choose t which is polynomial in d and n. Expansions properties of the graphs $G[v]$ are clearly relevant.

2. *Other randomized pivot rules.* A large class of randomized rules $\mathcal{D}(r)$ where $r = r(d, n)$ is a function of d and n, such that $r(d, n)$ is an integer between 1 and $d - 1$ was suggested by the author in [15]. Given an LP problem of dimension d with n constraints, and a feasible vertex v, **choose an r-face F containing v at random, apply the algorithm recursively to find the top vertex w of F, set $v =: w$ and repeat.**

If we choose $r(d,n) = 1$ we go back to random pivoting $\mathcal{A}$.

It was suggested in [15] to choose $r(d,n) = d(1 - \lceil d/n \rceil)$. Denote the resulting algorithm by $\mathcal{E}$. There are some heuristic reasons to believe that the number of steps in the external loop of $\mathcal{E}$ is $O(logn(n/d))$. (For this choice of r, given an oracle which will make the best possible choices of the r-faces F, the number of steps in the external loop will be at most $2(n/d)$. This was the proof of the first subexponential upperbound on $H(d,n)$.) Proving this, would imply that $\mathcal{E}$ is subexponential. More significantly this will give a quasi-polynomial strong algorithm for LP (by simultaneous applications on problems and their duals). See [15]. It make sense also to consider a relaxation of $\mathcal{E}$ using the random walk idea of algorithm $\mathcal{B}$.

If we choose in algorithm $\mathcal{D}$ $r(d,n) = d - 1$ we get the algorithm $\mathcal{S}_0$ (random antipivoting) which was considered in Section 3.

3. One may replace the random choices in the algorithms described above by some deterministic rules. Dealing with Algorithm $\mathcal{S}_0$ this can be called an *antipivot rule.* In each step given a vertex v we choose a direction (antipivot) and then we ignore it completely until solving the problem for the facet determined by the other directions. We can choose the antipivot by reversing some of the common pivot rules. Is there a deterministic antipivoting which is subexponential? Is this a good idea in practice? Similarly one may consider choosing r directions and staying in the r-face determined by them (via recursive application of the algorithm) until the top (or according to some other criterion) before doing anything else. As a matter of fact, similar algorithms exist in practice.

Günter Ziegler pointed out the relations with pivot rules suggested by Bland [2] (see also Chapter 10 of [1]). The class of algorithms obtained by all possible antipivoting rules is equivalent to the class of algorithms called "Algorithm A" by Bland. So algorithm $\mathcal{S}_0$ is an implementation of Bland's Algorithm A in which all choices of the algorithm are made at random. (Note that "Algorithm A" is much more general than Bland's famous "smallest subscript" algorithm which is defined in the same paper.)

4. For a description of Clarkson's remarkable algorithm and why it works the reader is advised to rush to [6].

5. Seidel's algorithm [29] goes as follows: **Choose at random a constraint and optimize (using the algorithm recursively) on the hyperplane determined by the constraint. If the outcome is not the solution for the entire problem delete the constraint and continue.** This is a self-dual algorithm, and the path it describes on the graph of the hyperplane arrangement contains vertices which are neither feasible nor dual-feasible. (With an oracle which makes the best random choices d steps suffice.)

In our algorithms at each step we can choose a random constraint only from a restricted subset of the set of all constraints, but we can gain quite a lot even if the optimum on the hyperplane determined by the constraint is not the global optimum.

6. It would be interesting to find a better algorithm even for the very special case when the feasible polyhedron in combinatorially equivalent to the d-dimensional cube. In this case from every vertex there is a monotone path of length d to the top.

7. It can be shown (using the Milnor-Thom theorem like in [12]) that the number of directed graphs obtained from problems in $\mathcal{L}(d,n)$ is at most $2^{P(d,n)}$, where P is a polynomial. It follows via a standard argument that there is a (deterministic) arithmetic circuit of subexponential size for all problems in $\mathcal{L}(d,n)$.

8. It is worthwhile to mention that there is a substantial knowledge on the behavior of the simplex algorithm (with respect to a certain class of pivot rules, first introduces by Gass and Saaty) for a "random" linear programming problem. It is known for several probability distributions on $\mathcal{L}(d,n)$ that the average number of steps of the simplex algorithm (for these pivot rules) is polynomial in d and n. The first results in this direction were made by Borgwardt, and Smale. We refer the reader to Borgwardt's book [3] for a description of his work and for references to the work of Smale, Megiddo, Haimovitch, Adler and Megiddo, Adler, Karp and Shamir, Todd and others.

Although all pivot rules for which a polynomial behavior on "average" problems was proved are related to the Gass-Saaty rule, it is not known if there exists *any* pivot rule which is not polynomial on an average problem. In other words for $O \in \mathcal{L}(d,n)$, let $\gamma(O)$ denotes the length of the maximal monotone path in the feasible polyhedra of O. Is the expected value of $\gamma(O)$ polynomial? (Here O is taken at random and the reader may choose his favorite probabilistic model.)

References

[1] A. Björner, M. Las Vergnas, B. Sturmfels, N. White and G. M. Ziegler, Oriented Matroids, Cambridge University Press, 1992, to appear.

[2] R. G. Bland, New finite pivoting rules for the simplex method, Math. of Op. Res. 2(1977), 103-107.

[3] K. H. Borgwardt, The Simplex Method, a Probabilistic Analysis, Algorithms and Combinatorics 1, Springer-Verlag, Berlin, 1987.

[4] V. Chvátal, Linear Programming, W. H. Freeman, San Francisco, 1983.

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

[6] K. L. Clarkson, A las Vegas Algorithm for linear programming when the dimension is small, Proceedings STOC 1988, 452-456.

[7] G. B. Dantzig, Maximization of a linear function of variables subject to linear inequalities, in: Activity Analysis of Production and Allocation, T. C. Koopman (Ed.) Cowles Commission Monograph 13, Wiley, New York, 1951, 339-347.

[8] G. B. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, N.J., 1963.

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

[10] M. E. Dyer and A. Frieze, random walks, totally unimodular matrices and a randomized dual simplex algorithm, 1992, to appear.

[11] D. Gale, How to solve linear inequalities, Amer. Math. Monthly 76(1969),589-599.

[12] J. E. Goodman and R. Pollack, Upper bounds for configurations and points in R^d, Discrete and Comp. Geometry, 1(1986), 219-227.

[13] B. Grünbaum, Convex Polytopes Ch. 16, Wiley Interscience, London, 1967.

[14] G. Kalai, The diameter problem for convex polytopes and f-vector theory, in: The Victor Klee Festschrift (P. Gritzman and B. Sturmfels, eds.), pp. 387-411, Amer. Math. Society, Providence, 1991.

[15] G. Kalai, Upper bounds for the diameter of graphs of convex polytopes, Discrete Comp. Geometry 7(1992), to appear.

[16] G. Kalai and D. J. Kleitman, A quasi-polynomial bound for diameter of graphs of polyhedra, Bull. Amer. Math Soc., 24 (Apr. 1992), in press.

[17] N. Karmarkar, A new polynomial time algorithm for linear programming, Combinatorica, 4 (1984), 373-397.

[18] L. G. Khachiyan, A polynomial algorithm in linear programming, Soviet Math. Doklady, 20 (1979), 191-194.

[19] V. Klee, Convex polyhedra and mathematical programming, Proc. 1974 Int. Congress math. Vancouver, 51-56.

[20] V. Klee and P. Kleinschmidt, The d-steps conjecture and its relatives, Math. of Oper. Res. 12(1987), 718-755.

[21] V. Klee and G.J. Minty, How good is the simplex algorithm, in: Inequalities III, pp. , O. Shisha (ed.) Academic Press, New-York ,1972, pp. 159-175.

[22] V. Klee and D. Walkup, The d-step conjecture for polyhedra of dimension $d < 6$, Acta Math., 133, 53-78.

[23] D. G. Larman, Paths on polytopes, Proc. Lon. Math. Soc. 20(1970), 161-178.

[24] J. Matoušek, M. Sharir and E. Welzl, A subexponential bound for linear programming, [illegible] 8-th Annual Symp. on Computetional Geom[illegible] 1992, to appear.

[25] P. McMullen, The maximal number of faces of a convex polytope, Mathematika 17(1970), 179-184.

[26] N. Megiddo, Toward a genuinely polynomial algorithm for linear programming, SIAM J. Comp. 12(1983), 347-353.

[27] N. Megiddo, Linear programming in linear time when the dimension is fixed, J. Assoc. Com[illegible] Mach. 31(1984), 114-127.

[28] A. Schrijver, Theory of Linear and Integer Programming, Wiley-Interscience 1986.

[29] R. Seidel, Linear programming and convex hulls made easy, in "Proc. 6th Annual ACM Symposium on Computational Geometry" (1990), 211-215.

[30] M. Sharir and E. Welzl, A Combinatorial bound for linear programming and related problems, Proc. 9-th Symp. Theor. Aspects of Computer Science (1992), Lecture Notes in Comp. Sci. 577, pp. 569-579, to appear.

[31] E. Tardos, A strongly polynomial algorithm to solve combinatorial linear programs, Oper. Res., 34(1986), 250-256.