SIGACT News 26 no. 2, pp. 96–104, 1995

A Survey of Linear Programming in Randomized Subexponential Time

Michael Goldwasser *

Abstract

"It is remarkable to see how different paths have led to rather similar results so close in time." – Kalai, 1992 ([8]).

Three papers were published in 1992, each providing a combinatorial, randomized algorithm solving linear programming in subexponential expected time. Bounds on independent algorithms were proven, one by Kalai, and the other by Matoušek, Sharir, and Welzl. Results by Gärtner combined techniques from these papers to solve a much more general optimization problem in similar time bounds.

Although the algorithms by Kalai and Sharir–Welzl seem remarkably different in style and evolution, this paper demonstrates that one of the variants of Kalai's algorithm is identical (although dual) to the algorithm of Sharir–Welzl. Also the implication of Gärtner's framework on future improvements is examined more carefully.

1 Introduction

Linear programming has long been an important problem in computer science. Since 1950, when the simplex method was introduced by Dantzig [4], thousands of papers have been published regarding linear programming. The simplex method is a simple combinatorial algorithm widely used (and praised) in practice, however its analysis in respect to theoretical efficiency turned out to be rather difficult. Although many felt that the simplex method was polynomial in the worst case, Klee and Minty provided counterexamples in 1972, requiring exponential time for a common pivot rule [13]. The first polynomial algorithms for linear programming were given by Khachiyan [12] and Karmarkar [11] in the 1980's, however these algorithms are not strongly polynomial, that is their running times depended on the size of the coefficients representing the constraints, and not solely on n, the number of constraints, and d the number of variables. For linear programs in fixed dimensions, Megiddo (1983) was first to give a deterministic algorithm whose dependence on n is linear [16]. The dependence on d was further improved in 1986 by Dyer [5] and then Clarkson [2].

Clarkson used random sampling techniques in 1988 to develop a simple algorithm which solves linear programming by solving $O(d^2 \log n)$ smaller linear programs with $O(d^2)$ constraints and d variables [3]. By using a standard simplex algorithm as a subroutine for solving these small programs, Clarkson's algorithm is still exponential, however the best current bounds are realized by using one of these recent subexponential algorithms as a subroutine. Seidel introduced a simple, randomized, incremental algorithm in 1991 [19] which has an expected running time of $O(d!n)$. In 1992, Sharir and Welzl made a simple improvement on this algorithm [20] and proved an expected running

*Department of Computer Science, Stanford University, Stanford, CA, 94305. E-mail: wass@cs.stanford.edu.
Research supported by NSF Grant CCR-9215219 and by NSF Young Investigator Award CCR-9357849 with matching funds from IBM, the Schlumberger Foundation, the Shell Foundation, and the Xerox Corporation.

time of $O(d^3 2^d n)$. This algorithm was presented in a generalized framework that included linear programming and several other optimization problems.

Stemming from the study of the diameter of polytopes, Kalai developed a randomized simplex algorithm in 1992, and proved the first subexponential bound for linear programming [8]. Following this, Matoušek joined Sharir and Welzl to tighten the analysis and to prove that their previous algorithm, unchanged, also had a subexponential expected running time for linear programming [15]. However this bound does not necessarily hold for other optimization problems in their framework. Gärtner then combined a technique used by Kalai with the Sharir–Welzl algorithm, to provide another subexponential algorithm [6]. More importantly this algorithm was designed in a much more general framework of optimization problems, and thereby showed that few of the specific properties of linear programming were necessary in proving the subexponential running time. By using one of these subexponential algorithms as a subroutine in Clarkson's method, the best current bound of $O(d^2 n + e^{O(\sqrt{d \log d})})$ is achieved.

2 Linear Programming

We refer to the linear programming problem (LP) in the following standard form. Given variables $x = (x_1, \ldots, x_d)$, and constraints $a_i \cdot x \leq b_i$ for $i = (1 \ldots n)$, find x which maximizes the value $\{ c \cdot x \mid Ax \leq b \}$. Geometrically, this is equivalent to finding a vertex x^* extreme in some direction c within the polyhedron P defined by the intersection of a set H of n closed halfspaces in R^d. The reader is refered to [18] for a detailed discussion of linear programming theory, background, and recent results. Also [1] provides an excellent introduction to linear programming and the simplex method. The recent book [17] provides a comprehensive discussion of the analysis of randomized algorithms, including specific analysis of several of the LP algorithms discussed here.

Throughout this paper, we will make several assumptions which can be removed by standard techniques. Unless specifically mentioned elsewhere, for any input program we assume the polyhedron is both feasible and bounded, and we assume there exists a unique optimal solution. We assume that we have access to some feasible vertex (basis) when necessary. We assume non-degeneracy, in the sense that every vertex (basis) is incident to exactly d facets.

To be consistent with the notation of other papers, we assume w.l.o.g. that the objective function $c = (-1, 0, \ldots, 0)$, and therefore the goal is to minimize the value of a solution. We denote as $w(H)$ the value of the optimal solution. Given a subset $G \subseteq H$ of constraints, $w(G)$ is the optimal value of the linear program defined by only considering those constraints in G. Clearly $w(G) \leq w(H)$. If P is empty, we say $w(H) = \infty$, and if P is non-empty but unbounded in the objective direction we say $w(H) = -\infty$. A basis is a set of constraints, B, such that $w(B) > -\infty$ and $w(B) \geq w(B')$ for any $B' \subset B$. A basis of H is a minimal subset $B \subseteq H$ with $w(B) = w(H)$. The goal is to find a basis of H.

3 Evolution of Subexponential Randomized Algorithms

3.1 Clarkson's Algorithm

Clarkson's recursive algorithm relies on this important observation. If we assume the optimum is unique, then there must exist a basis B of H with d or fewer constraints. Given any subset $G \subseteq H$ of constraints, solving the relaxed linear program based on G will have one of two results. If G contains B, then the optimum of G will equal the optimal of the full set H. If not, then there must

be at least one constraint in B which is violated by this solution of G. Clarkson's goal is to find a set G which contains the basis, but which has a fairly small size compared to the original set H.

Such a small set G is built up as follows. Starting with G empty, a subset $R \subset H \setminus G$ is chosen at random with $|R| = d\sqrt{n}$. The optimum of the linear program $(R \cup G)$ is found recursively, and if this solution does not violate any of the constraints in H then it must also be the optimum of H, the overall goal. Otherwise at least one constraint of the goal basis must be violated by the current solution, and that constraint was not yet in G since this solution respects $(R \cup G)$. Adding all violated constraints into the set G, assures that G will contain one more constraint of the final basis. Since there are at most d such constraints, after d passes through this loop, the entire basis will be contained in G and thus the recursive call will provide the true optimum.

The key to the analysis is that by choosing R at random, the expected number of violated constraints at each pass is around $\sqrt{n}$ and so the set G can be built with size $O(d\sqrt{n})$. As a base case to this recursive algorithm, if there are ever less than $9d^2$ constraints, the optimum is found using a simple algorithm as a subroutine. The analysis of a more refined version of this idea bound the number of calls to the base subroutine to be $O(d^2 \log n)$. Using a simplex algorithm here still results in an exponential time bound, however if a more efficient procedure can be used in place of the subroutine, then the performance of this algorithm immediately improves.

3.2 Seidel's Algorithm

Seidel's algorithm is an amazingingly simple randomized, incremental algorithm for linear programming, and the analysis is just as clean. The basic algorithm is as follows: Pick a random constraint $h \in H$. Recursively find the optimal vertex v in $H - h$. Now if h does not violate v, then it is indeed the optimal solution for the original problem. If it does violate v, then the true optimal solution must lie on h, so cast all the constraints onto their intersection with h and solve recursively. This subproblem has its dimension and number of constraints each reduced by one. (Note: this algorithm can also be thought of as starting from scratch and incrementally adding in constraints maintaining the optimal solution for the currently seen subset.)

The key to the analysis is that every vertex is defined by only d constraints, so the chance of h being one of those is d/n. Even without the non-degeneracy assumption, the analysis still holds. Call a constraint $h \in H$ *extreme* if $w(H - h) < w(H)$. It is easy to show that even in a degenerate case, there can be at most d extreme constraints at a vertex. This leads to the recurrence, $T(d, n) \leq T(d, n-1) + O(d) + \frac{d}{n}(O(dn) + T(d-1, n-1))$. From this, the $O(d!n)$ upper bound can be shown inductively.

3.3 Matoušek/Sharir/Welzl Algorithm

A key flaw in Seidel's algorithm is that it throws away information. After finding the optimal vertex v of $(H - h)$, if h violates v, it solves the subproblem constrained onto h from scratch. The candidate vertex v is thrown away, even though it seems reasonable that constraints defining v are more likely to also define the true optimal vertex. This idea is brought out by the algorithm Matoušek, Sharir, Welzl [20] [15].

To maintain extra information, they add a second parameter to the algorithm. The LP program is called with two sets, the set of constraints H, and a basis $C \subseteq H$. Note that this basis is not necessarily a basis of H, in fact the overall goal is to find a basis of H. Whereas Seidel's algorithm incrementally adds in constraints from H in random order, this algorithm automatically starts with the constraints in C and then adds in all elements of $H - C$ in random order.

This subtle change allows them to prove the subexponential bound. Recall, a constraint is extreme

in H if $w(H-h) < w(H)$, and there are at most d such constraints. However if some of the extreme constraints are in the parameter C, then the probability that a random constraint of $H-C$ is extreme is even less. We call a constraint h *enforcing* in basis $B \subseteq H$, if $w(H-h) < w(B)$. This means that not only must h be in B, but h must also be in any basis with value greater than $w(B)$. Since this algorithm walks through a set of candidate bases with non-decreasing values, if a constraint is enforcing in the current candidate basis B, then h will be enforcing in all future candidate bases.

There are at most d extreme constraints in H, and since all enforcing constraints in C are extreme in H, there can be at most $\Delta = (d - \text{the number of enforcing constraints})$ extreme constraints left in $H-C$. They denote Δ as the *hidden dimension*, and point out that the probability that a random constraint from $H-C$ is extreme is at most $\frac{\Delta}{|H-C|}$. (Compare this to the bound of $\frac{d}{n}$ from Seidel's algorithm).

After choosing a random constraint $h \in H-C$, they recursively find a basis $B_{(H-h)}$ of $(H-h)$. Again, if h does not violate this basis, then it is also a basis for H. If h violates $B_{(H-h)}$ then they set the parameter $C = \text{Basis}(B_{(H-h)} \cup \{h\})$. Since $w(H-h) < w(C)$, then h is enforcing in C, reducing the hidden dimension of the subproblem by one. More importantly, the randomization suggest that the hidden dimension reduces by even more.

Consider all constraints $\{e_1, \ldots, e_t\} \subset H-C$ which are extreme in H, and enumerate them such that $w(H-e_1) \leq w(H-e_2) \leq \cdots$. If a random constraint chosen from $H-C$ violates $B_{(H-h)}$, then it is equally likely to be any e_i. The above discussion explained that e_i will be enforcing in C, however for $1 \leq j \leq i$, $w(H-e_j) \leq w(H-e_i) < w(C)$, and so e_j will also be enforcing in C. Therefore the hidden dimension of the problem will always be reduced by i, where i is chosen uniformly at random from $(1 \ldots t)$.

This analysis develops into a simple recurrence for $f(k,n)$, bounding the number of basic operations for a problem with n constraints, and hidden dimension $\leq k$. The preceding discussion leads to the recurrence, $f(k,n) = f(k,n-1) + \frac{1}{n}\sum_{i=1}^{\min(n,k)} f(k-i,n)$, for $k,n \geq 1$. The solution to this results in the subexponential bound,

$$f(k,n) \leq min\left\{O(2^k n), \exp\left(2\sqrt{k \ln(\frac{n}{\sqrt{k}})} + O(\sqrt{k} + \ln n)\right)\right\}$$

They prove a lower bound on $f(k,n)$ which almost matches this [15], showing that their solution to this recurrence is tight. Of course this does not necessarily prove a lower bound on the running time of the algorithm on linear programming, since different analysis may provide a better recurrence. However more recent results in [14] discuss lower bounds for this algorithm.

3.4 Matoušek/Sharir/Welzl Framework

The algorithm in [15] is actually presented in a more general framework that includes LP. They define *LP-type* problems as follows. The input to the problem is a pair (H,w), where H is a finite set called *constraints*, and $w : 2^H \to \mathcal{W}$ is a function mapping a subset of H into the linear order $(\mathcal{W}, \leq)$. The order $\mathcal{W}$ is assumed to have an element $-\infty$ which acts as a minimum value. A basis is defined in a similar manner as for linear programming. A problem is *basis-regular* if every basis has the same cardinality, and that cardinality is referred to as the *combinatorial dimension*.

The only axioms assumed about the linear order are the following:

- (Monotonicity) For any $F \subseteq G \subseteq H$, $w(F) \leq w(G)$.
- (Locality) For $F \subseteq G \subseteq H$ with $w(F) = w(G) > -\infty$, and $h \in H$, then $w(G) < w(G \cup \{h\})$ iff $w(F) < w(F \cup \{h\})$.

The algorithm requires two operations. Given a basis B and a constraint h, a *violation test* determines whether h is violated by B. Also, given a basis B and a constraint h, a *basis computation* computes a basis for $(B \cup \{h\})$. The bounds in the previous section hold for any basis-regular LP-type problem. Also, the results in [14] provide an example of an LP-type problem of combinatorial dimension d with $2d$ constraints for which their algorithm requires $\Omega(e^{\sqrt{2d}}/\sqrt[4]{d})$. This lower bounds the running time of their algorithm on LP-type problems. Of course this does not provide a lower bound for the running time of the algorithm on LP, since there may be properties of LP other than the above axioms which limit the running time.

3.5 Kalai's Algorithm

The algorithm of Kalai [8] emerged from the study of the diameter problem for graphs of polyhedra. Given a linear program O, and its polyhedron P, the graph $G(P)$ is the one-shell of the polyhedron. Part of the importance of this graph is that a pivot in the simplex method corresponds to following one of the edges in $G(P)$. In fact, since a simplex method must always increase the objective function at each step, we can direct the edges in $G(P)$ so that they point from lower to higher value. Denote this graph as $\overline{G}(O)$.

Let $\Delta(d, n)$ denote the maximum diameter over all possible polyhedra with n facets in d dimension. The value of $\Delta(d, n)$ provides an automatic lower bound for the simplex method with any pivot rule. If there exists a polyhedron so that two vertices have shortest path $\Delta(d, n)$, then by forcing the algorithm to start at one vertex and reach the other, any series of pivots will take at least $\Delta(d, n)$ steps.

A superpolynomial lower bound on $\Delta(d, n)$ would eliminate the possibility of a polynomial pivot rule, however there is a long-running conjecture of Hirsch [4] that $\Delta(d, n) = O(n + d)$ for bounded polyhedra. Until recently though, the best upper bounds had been exponential. Kalai gave the first subexponential bound [9], and this bound was improved by Kalai and Kleitman [10] to prove $\Delta(d, n) = O(n^{\log d + 2})$. These proofs led to the development of the subexponential algorithm in [8].

Before describing the algorithm we define the concept of an *active* facet. A facet F is *active* with respect to a vertex v, if $\max\{\phi(x) : x \in F\} > \phi(v)$, where $\phi(x)$ denotes the value of the objective function at point x. There are several variants of the algorithm, but they describe the general algorithm as follows.

- Starting from v, reach vertices in at least $r(n, d)$ active facets.
- Choose a facet F at random among these r facets, and apply the algorithm recursively to reach an optimal vertex w in F. Set $v =: w$ and repeat first step.

Of course there are two key issues with the above description: How to choose the value of the parameter $r(n, d)$, and then how to efficiently find a set S of $r = |S|$ active facets. If $r = d$, then finding the set S is easy, since the vertex v is incident to d facets. In a non-degenerate situation, if v is not optimal, then at least $d - 1$ of the incedent facets will be active. If one is inactive, then there will exists a unique pivot step away from this facet which will lead to a d^{th} active facet. The algorithm obtained by setting $r = d$ is referred to as variant $\mathcal{S}_0$, and will be discussed more in section 4.

Kalai's best results came by setting $r = \max(d, n/2)$. In the case that $n/2 \geq d$, finding the first d facets to put in S is done as above. To find another facet, they construct the linear program using only the constraints currently in S along with any inactive facets incident to v. Call this linear program O' and its polyhedron P'. Clearly v is a feasible vertex of P' since all constraints incident to v are in the program. By starting to solve this program, one of two things will happen. If the entire path taken in P' is inside the feasible region of the original polyhedron, then the optimal solution to

O' will be the true optimal solution for the original problem. If this is not the case, then at some point the path in P' leaves the original feasible region. The very first edge leaving the original feasible region must actually be crossing some facet of P not in P', and so this facet can be added to S. Therefore, by solving a linear program with d constraints, the $(d+1)^{st}$ facet is added to S. This process can be repeated until r facets are found. If $g(d,n)$ is the time to solve a linear program with n facets in d dimensions, then building up S in this manner will take time, $\sum_{i=d}^{n/2} g(d,i)$.

The key to the analysis of this algorithm is based on counting the active facets. Since any simplex method traces through vertices with non-decreasing values, a path starting at v can never pass through inactive facets. Clearly all facets placed into the set S in the algorithm are active, since they are reachable by way of non-decreasing steps. Now enumerate all facets in $S = \{F_1, F_2, \ldots, F_r\}$, such that $\max\{\phi(x) : x \in F_i\} \leq \max\{\phi(x) : x \in F_{i+1}\}$. By picking a random facet in S and finding the optimal vertex on that facet, i facets become inactive, where i is chosen uniformly from $(1, \ldots, r)$. Although there are many variants of this algorithm, this leads to the general recursion,

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

Analyzing this recurrence leads to the subexponential results [8].

3.6 Gärtner's Framework

A powerful result was provided by Gärtner when he created a very generalized framework including LP, and displayed that the above algorithms could be adapted to produce subexponential expected running time in this framework [6]. He defines an *Abstract Optimization Problem* (AOP) as a triple $(H, <, \phi)$, where H is a finite set, $<$ is a linear order on 2^H, and ϕ an oracle. For any given $F \subseteq G \subseteq H$, the oracle will determine if $F = min(2^G)$ and if not, returns a witness set with smaller value. The goal of the problem is to find $min(2^H)$. In terms of the framework in [15], the linear order is exactly $\mathcal{W}$, and the oracle can be efficiently implemented if F is a basis by simply searching for a violating constraint, and if one exists, making a basis computation by adding that constraint.

For a complete description of the algorithm, the reader is referred to [6], however the general algorithm is modeled after the algorithm of Sharir–Welzl, borrowing one trick from Kalai. The basic algorithm starts at some candidate F and follows these steps:

- if $F = min(2^H)$ then return F.
- if not, choose a random element $h \in H - F$ and compute $F' = min(F, H - \{h\})$.
- If $\phi(F') = min(2^H)$, then return F', else set $F =: \phi(F')$ and start again.

So far this is exactly the same as the algorithm of [20]. The efficiency of the randomization relies on the fact that there are enough choices in $H - F$. In [20], they rely on the basis-regular property of LP, and so every candidate F they encounter has cardinality exactly d. The only time a problem arises is when H is not much bigger than F, however in this case, there are only a small number of bases to check, so even a brute force solution will not take very long.

However, this fact falls apart in the AOP framework since there are no assumptions made about the candidate subsets F. Therefore, Gärtner borrows a trick from [8]. If the set of candidates, $S = H - F$ is determined to be too small, he expands it. He starts running the algorithm on a subproblem until it finds the overall optimal $min(w^H)$, or it provides a new candidate that was not previously in S. By repeating this process, the set S can be expanded as desired leaving enough candidates for the randomization to be effective.

The analysis is based on what he terms *freedom*, a concept very similar to the active facets of [8] or enforcing constraints of [15]. This results in a recurrence quite similar to that of [8].

4 Duality of Algorithms

In this section we consider the implications of the duality theorem of linear programming. More specifically, we show that the algorithm of Sharir–Welzl and variant $\mathcal{S}_0$ of Kalai's algorithm are exactly dual to each other. The relation between these two algorithms was suggested by Kalai in [8], however the exactness of the parallelism was not noted.

From Section 2 our goal is to solve the linear program $\max\{\, cx \mid Ax \leq b \,\}$. By the duality theorem of linear programming, the dual of this program is $\min\{\, yb \mid yA = c, y_i \geq 0 \,\}$. We know that the optimal solutions to these programs have equal values, and given an optimal solution to one it is easy to construct an optimal solution to the other. A basis in the primal problem corresponds exactly to a vertex in the dual polytope (and vice versa). More importantly, if a constraint is thrown out from the primal, the effect on the dual is to force the corresponding non-negativity constraint to be tight.

We will now walk through the steps of the Sharir–Welzl algorithm, while simultaneously "translating" to the terminology of the dual program.

Sharir–Welzl Algorithm	
Primal	Dual
$\max cx$ $a_i x \leq b_i \quad (i = 1 \ldots n)$	$\min yb$ $y_i \geq 0 \quad (i = 1 \ldots n)$ $ya_j = c_j \quad (j = 1 \ldots d)$
$C \leftarrow$ an initial feasible basis	$C \leftarrow$ an initial feasible vertex
Choose a random $h \in H - C$	Choose a random y_h of all $y_h = 0$. (i.e., choose random facet incident to C)
Solve $(H - h)$ using C as start point getting resulting basis B	Solve on $y_h = 0$ facet with C as start vertex getting resulting vertex B
If h violates B then $C' \leftarrow \text{basis}(B, h)$	If B not optimal vertex then $C' \leftarrow$ pivot B away from $(y_h = 0)$ facet.
Restart from C'	Restart from C'

From the above table, it becomes clear that the Sharir–Welzl algorithm is a pure dual-simplex algorithm. In fact, the algorithm is almost exactly variant $\mathcal{S}_0$ of Kalai's algorithm run on the dual polytope. The only step of the parallelism which is obscure is step four in the above table, and the obscurity is due the following apparent lapse in the description of the $\mathcal{S}_0$ algorithm in [8].

It is evident through duality throwing out a random constraint of $H - C$ is identical to tightening a random constraint in the dual, of those incident to the current vertex. The only question is how the two algorithms continue once the subproblem is solved.

A key point in the definitions of *enforcing* constraints and *active* facets is that the current solution must be *strictly* greater than the best possible solution under the extra conditions. That is, h is not enforcing in $B_{(H-h)}$ since $w(H - h) \not< w(B_{(H-h)})$ but it will be enforcing as soon as any better candidate is found. For this reason, they calculate $\text{basis}(B_{(H-h)}, h)$, and from that point on h is indeed enforcing, and the analysis holds.

In terms of active facets, if B is the optimal vertex on a given facet, then that facet is not active with respect to B. In this case, there exists only one edge leaving B that improves the objective function, and that edge is taken by pivoting away from the inactive facet. Therefore any simplex

method must take that edge once at B. Variant $\mathcal{S}_0$ in [8] is defined as the algorithm resulting when parameter $r = d$. However it was loosely stated that this algorithm simply picks one of the d facets incident to a vertex. Of course in the case mentioned here, only $d-1$ of those constraints are active, and so one more must be found. However the first facet found by any walk will be exactly the one introduced when the inactive facet is pivoted out. Therefore the bag of d random choices used by both algorithms are identical, and hence the algorithms are exactly dual.

This relation between the algorithms corroborates an interesting point in the analysis of [8]. For the most efficient version of their algorithm, they prove subexponential bounds on $f(d,n)$, $f(d,2d)$, and $f(d,d+m)$, however for variant $\mathcal{S}_0$, they could only bound $f_0(d,d+m)$. As they point out, $f(d,d+m)$ can be viewed as a bound on running the algorithm on the dual polytope, which has m variables, and $m+d$ constraints. Although they could not bound the running time of variant $\mathcal{S}_0$ on the primal polytope, they prove that the variant is indeed subexponential when run on the dual polytope. That gives a direct bound on the running time of the Sharir–Welzl algorithm, and most recent bound given in [15] matches this, with the exception of the constant in the exponent.

5 Implications of AOP results

The amazing realization that falls out of Gärtner's result [6] is that little of the structure of linear programming is being relied on to prove the subexponential bound on the number of oracle calls. Given any linear program, it is simple to translate the values of all the bases to create an instance of AOP. However very few of the $(2^n)!$ possible AOP linear orders are realizable as an LP problem. Of course, the structure of LP is being used in the implementation of the oracle, yet the bound on the number of oracle calls does not use any other axiom. Constructing a good classification for which orders are realizable may be very instructive in developing a better LP algorithm, however describing such a classification becomes quite difficult.

Also, Developing a non-trivial randomized lower bound for AOP would be a useful result. It is quite possible that a superpolynomial lower bound can be constructed for AOP, in fact any progress in closing the gap between the upper bound of [6] would be useful. Such a lower bound would probably not apply to a linear order realizable as an LP program, and so developing such a lower bound would be instructive in improving a specialized linear program algorithm.

Another important open area is to better analyze the simple randomized simplex rule suggested by Dantzig. That is, given the current vertex, simply choose a pivot rule uniformly at random from all incident edges which improve the objective function. Gärtner and Ziegler have recently analyzed this algorithm's performance of the Klee-Minty cubes [7], and established quadratic upper bounds on the expected number of steps for this instance.

The expected time for this algorithm can be modeled as the time it takes for a random walk on a directed acyclic graph to reach a unique sink. The key question is what properties of the graph are necessary to give reasonable upper or lower bounds on the maximum over all starting vertices. Clearly, the height of a polytope studied in [8], is an important property, however other properties might also prove valuable in bounding the complexity.

Acknowledgments: The author wishes to thank Rajeev Motwani, Leo Guibas and Bernd Gärtner for their helpful discussions.

References

[1] V. Chvátal. *Linear Programming*. W. H. Freeman, New York, NY, 1983.

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

[3] K. L. Clarkson. A Las Vegas algorithm for linear programming when the dimension is small. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 452–456, 1988.

[4] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

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

[6] B. Gärtner. A subexponential algorithm for abstract optimization problems. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 464–472, 1992.

[7] B. Gärtner and G. Ziegler. Randomized simplex algorithms on klee-minty cubes. In *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 502–510, 1994.

[8] G. Kalai. A subexponential randomized simplex algorithm. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 475–482, 1992.

[9] G. Kalai. Upper bounds for the diameter of graphs of convex polygopes. *Discrete Comput. Geom.*, 7:to appear, 1992.

[10] G. Kalai and D. J. Kleitman. A quasi-polynomial bound for diameter of graphs of polyhedra. *Bulletin of the American Math. Soc.*, 24:315–316, 1992.

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

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

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

[14] J. Matoušek. Lower bound for a subexponential optimization algorithm. Technical Report B-92-15, Fachbereich Mathematik, Freie Universität Berlin, Berlin, 1992.

[15] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 1–8, 1992.

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

[17] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

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

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

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