Las Vegas Algorithms for Linear and Integer Programming When the Dimension is Small

Kenneth L. Clarkson
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Abstract

This paper gives an algorithm for solving linear programming problems. For a problem with n constraints and d variables, the algorithm requires an expected

$$O(d^2 n) + (\log n) O(d)^{d/2+O(1)} + O(d^4 \sqrt{n} \log n)$$

arithmetic operations, as $n \to \infty$. The constant factors do not depend on d. Also, an algorithm is given for integer linear programming. Let φ bound the number of bits required to specify the rational numbers defining an input constraint or the objective function vector. Let n and d be as before. Then the algorithm requires expected

$$O(2^d dn + 8^d d\sqrt{n \ln n} \ln n) + d^{O(d)} \varphi \ln n$$

operations on numbers with $d^{O(1)}\varphi$ bits, as $n \to \infty$, where the constant factors do not depend on d or φ. The expectations are with respect to the random choices made by the algorithms, and the bounds hold for any given input. The technique can be extended to other convex programming problems. For example, an algorithm for finding the smallest sphere enclosing a set of n points in E^d has the same time bound.

1 Introduction

In some applications of linear and quadratic programming, the number of variables will be small. Such applications include Chebyshev approximation, linear separability, and the smallest enclosing circle problem. Megiddo [Meg84] gave an algorithm for linear programming that requires $O(2^{2^d} n)$ time, where n is the number of constraints and d is the number of variables. (Unless otherwise indicated, we assume unit-cost arithmetic operations.) This time bound is optimal with respect to n, and acceptable when d is very small. Variant algorithms have been found with the slightly better time bound $O(3^{d^2}n)$ [Cla86, Dye86]. Unfortunately, Megiddo's approach must take $\Omega(d!n)$ time, since it recursively solves linear programming problems with fewer variables [Dye86]. Dyer and Frieze [DF89] used random sampling [Cla87, CS89] to obtain a variant of Megiddo's algorithm, with an expected time bound no better than $O(d^{3d}n)$. This paper gives an algorithm requiring expected time $O(d^2n) + (\log n)O(d)^{d/2+O(1)} + O(d^4\sqrt{n}\log n)$, as $n \to \infty$, where the constant factors do not depend on d. The leading term in the dependence on n is $O(d^2n)$, a considerable improvement in d. The second term arises from the solution by the algorithm of $O(d^2 \log n)$ "small" linear programs with $O(d^2)$ constraints and d variables. The solution of these linear programming problems with the simplex algorithm requires $O(d)^{d/2+O(1)}$ time. The third term is discussed in §3.

H. W. Lenstra was the first to show that integer linear programming problems can be solved in polynomial time when the number of variables is fixed [Len83]. His algorithm was subsequently improved ([Kan87, FT87], see also [Bab85, Fei84]), so that the fastest deterministic algorithm for this problem requires $d^{O(d)}n\varphi$ operations on $d^{O(1)}\varphi$-bit numbers. Here φ is the *facet complexity* (or *row complexity*) of the input, the maximum number of bits used to specify an input inequality constraint. (The value φ is required also to be larger than the number of bits specifying the objective function vector. This condition can be avoided using the techniques of [FT87].) The new algorithm requires expected

$$O(2^d n + 8^d\sqrt{n \ln n}\ln n)$$

row operations; such an operation is just the evaluation of an input inequality at a given integral point. The rows have no more than φ bits; the integral points can be specified with $7d^3\varphi$ bits. The algorithm also calls Lenstra's algorithm for several "small" integer programming problems. This gives the second term in the operation bound. When $n \gg d$, the new algorithm is substantially faster than Lenstra's.

The key idea of the algorithms is random sampling, applied as in [Cla87, CS89], to quickly throw out redundant constraints.

The next section presents the algorithm. In Section 3, a time bound is given and proven. The integer programming algorithm is described and analyzed in §4. The last section contains some concluding remarks.

2 Linear Programming

2.1 The problem

We will consider a specific form of the linear programming problem, and in this subsection, show that there is no generality lost in assuming that the problem has a unique solution.

Suppose a system of linear inequality constraints $Ax \leq b$ is given, where A is a given $n \times d$ matrix, b is a given n-vector, and x is a d-vector $(x_1, \ldots, x_d)$. Each inequality in this system defines a closed halfspace H of points that satisfy that inequality. The collection of these n halfspaces is a set S. The intersection $\cap_{H \in S} H$ is a polyhedral set $\mathcal{P}(S)$.

Consider the LP (Linear Programming) problem of determining the maximum x_1 coordinate of points x satisfying all the constraints, or

$$x_1^* = \max\{x_1 \mid Ax \leq b\}.$$

This is equivalent to a general version of the problem, using a change of coordinates. (For background on linear programming, see e.g. [Sch86].) For arbitrary S, this version of the problem may have no solutions: either $\mathcal{P}(S)$ is empty and the problem is infeasible, or x_1 can be arbitrarily large, and the problem is unbounded. Moreover, the problem may be bounded and feasible, but with many points in $\mathcal{P}(S)$ with the same maximal x_1 coordinate. It will be convenient for describing algorithms to change the problem slightly so that the LP problems given as input have exactly one solution.

First, the issue of feasibility. As is common, we'll split the LP problem into two phases: phase 1, finding a feasible point, and phase 2, finding the solution. As in [GMW91][§7.9.2] for example, the phase 1 problem can be solved by solving the LP

$$\max\{t \mid Ax + t1 \leq b, t \leq 0\},$$

where 1 is the n-vector of 1's. This problem is feasible, with a feasible point readily obtainable. If the optimum t is negative, the original problem is infeasible. Otherwise, we have a feasible point for the original problem.

Note that if we have a feasible point $x_0 \in \mathcal{P}(S)$ for an LP problem, the problem with constraints $A(x - x_0) \leq b - Ax_0$ has the origin 0 as a feasible point, and its solution y gives a solution $y + x_0$ to the original problem.

These considerations show that an algorithm for solving LP problems that have $0 \in \mathcal{P}(S)$ can be used to solve general LP problems; it will be clear that there is no asymptotic increase in complexity. Hereafter, we'll assume that an input LP problem has $0 \in \mathcal{P}(S)$, or equivalently $b \geq 0$. This will be useful in defining the solution to an unbounded problem.

If a given LP problem is bounded, we will use the minimum norm solution, that is, the point $x^*(S)$ with Euclidean norm $\|x^*(S)\|_2$ equal to

$$\min\{\|x\|_2 \mid x \in \mathcal{P}^*(S)\},$$

where $\mathcal{P}^*(S)$ is the convex polytope $\mathcal{P}(S) \cap \{x \mid x_1 = x_1^*\}$. A simplex-like algorithm to find such a solution is given in [GMW81][§5.3.3]. Note that the minimum-norm solution is unique: if u and v are two distinct minimum-norm solutions with $u \cdot u = v \cdot v = z$, then $c \equiv (u + v)/2 \in \mathcal{P}^*(S)$ and $c \cdot c < z$.

Thus we may assume that the given LP problem has $0 \in \mathcal{P}(S)$, and if bounded, has a unique solution. We can define a unique solution even when the given LP problem is unbounded, so that points with arbitrarily large x_1 coordinates are in $\mathcal{P}(S)$. Here the solution will be a ray with an endpoint of 0, in the direction of $x^*(\hat{S})$, where $\hat{S}$ is the set of constraints $Ax \leq 0$, together with the constraint $x_1 = 1$. Plainly this problem is bounded and has $0 \in \mathcal{P}(S)$; thus its unique solution gives an unbounded problem a unique solution ray.

Note that a ray $x^*(S) \subset \mathcal{P}(S)$, since $0 \in \mathcal{P}(S)$. Because of this, we will say that $x^*(S)$ satisfies all the constraints of S. In general, a ray z will be said to satisfy a constraint halfspace H just when $z \subset H$; otherwise z violates H.

We now have, with no loss of generality, a version of the linear programming problem that always has a unique optimum solution; finding such a solution can be done by a simplex or simplex-like algorithm. The algorithms described below will call such an algorithm for "small" subproblems, and thereby obtain solution points or rays.

2.2 The algorithm

Actually, four LP algorithms will be mentioned in this section, including the simplex-like algorithm, called by invoking a function $x^*_s(S)$. A recursive algorithm $x^*_r(S)$ will be introduced, and an iterative algorithm $x^*_i(S)$. Finally, a mixed algorithm $x^*_m(S)$ can be defined: it is a version of the recursive algorithm that uses the iterative algorithm for the recursive calls. The motivation for the mixed algorithm is to have a time bound with a leading term $O(d^2 n)$, while avoiding the larger number of calls to x^*_s of the recursive algorithm.

The recursive algorithm is based on the following facts: the optimum is unique, and is determined by some d or fewer constraints of S. That is, there is a set $S^* \subset S$ of size d or less such that $x^*(S^*) = x^*(S)$, so the optimum for S^* alone is the same as for S. The constraints in $S \setminus S^*$ are redundant, in the sense that their deletion from S does not affect the optimum.

The main idea of the recursive algorithm is the same as for Megiddo's algorithm: throw away redundant constraints quickly. The further development of this idea is very different, however. The algorithm builds a set $V^* \subset S$ over several phases. In each phase, a set $V \subset S \setminus V^*$ is added to V^*. The set V has two important properties: its size is no more than $2\sqrt{n}$, and it contains a constraint in S^*. After $d+1$ phases, V^* contains S^*, and also V^* has $O(d\sqrt{n})$ elements. That is, the algorithm quickly throws away the large set of redundant constraints $S \setminus V^*$. The algorithm proceeds recursively with V^*, and the recursion terminates for "small" sets of constraints. For these constraints, the appropriate optima are found using the simplex algorithm $x^*_s(S)$.

The algorithm is given in pseudo-code in Figure 1. The optimum $x^*(S)$ is computed as follows. Let $C_d = 9d^2$. If $n \leq C_d$, then compute the optimum $x^*(S)$ using the simplex algorithm. If $n > C_d$, then repeat the following, with V^* initially empty: let $R \subset S \setminus V^*$ be a random subset of size $r = d\sqrt{n}$, with all

```
function $x^*_r(S : set\_of\_halfspaces)$
return $x^* : LP\_optimum$;

$V^* \leftarrow \phi$; $C_d \leftarrow 9d^2$;
if $n \leq C_d$ then return $x^*_s(S)$;
repeat
  choose $R \subset S \setminus V^*$ at random, $|R| = r = d\sqrt{n}$;
  $x^* \leftarrow x^*_r(R \cup V^*)$;
  $V \leftarrow \{H \in S \mid x^*$ violates $H\}$
  if $|V| \leq 2\sqrt{n}$ then $V^* \leftarrow V^* \cup V$
until $V = \phi$;
return $x^*$ ;
end function $x^*$;
```

Figure 1: The randomized function x^*_r for LP.

subsets of that size equally likely. Let $x^* \leftarrow x^*(R \cup V^*)$, determined recursively, and let V be the set of constraints violated by x^*. If $|V| \leq 2\sqrt{n}$, then include V in V^*. (The value $2\sqrt{n}$ is twice the mean of $|V|$, which is $\approx dn/r$, as discussed in §3.) If V is empty, then exit the algorithm, returning x^* as $x^*(S)$. Otherwise, stay in the loop.

The iterative algorithm is based on a technique that will be termed *iterative reweighting*. (This was inspired by Welzl [Wel88], who applied the idea to halfspace range queries; a similar idea appeared in [Lit87], where it was applied to learning.) As in the recursive algorithm, a random subset is chosen, and also the set V of constraints violated by the optimum for that subset. However, we attach integer weights to the constraints, and choose random subsets by picking constraints with probabilities proportional to their weights. Since V contains a constraint in S^* (if nonempty) we know that the constraints in V are "important," and so increase their weights. This process is iterated, and eventually the constraints in S^* have large enough relative weight that the random subset we choose contains S^*.

The iterative algorithm is given in pseudo-code in Figure 2. Every constraint $H \in S$ has an integer weight w_H, initially one, and $w(V)$ denotes $\sum_{H \in V} w_H$ for $V \subset S$. The random subset R is chosen without replacement from the multiset corresponding to S, where the multiplicity of $H \in S$ is w_H. That is, we (conceptually) choose $H \in S$ with probability $w(H)/w(S)$, then set $w_H \leftarrow w_H - 1$, and repeat r times. (The resulting set R may be smaller than r, but this is actually an advantage.)

As noted above, the function $x^*_m(S)$ is simply $x^*_r(S)$, with the recursive call replaced by a call to $x^*_i(S)$.

```
**function** $x_i^*(S : set\_of\_halfspaces)$
**return** $x^* : LP\_optimum$;

**for** $H \in S$ **do** $w_H \leftarrow 1$ **od**; $C_d \leftarrow 9d^2$;
**if** $n \leq C_d$ **then return** $x_s^*(S)$;
**repeat**
  choose $R \subset S$ at random, $|R| = r = C_d$;
  $x^* \leftarrow x_s^*(R)$;
  $V \leftarrow \{H \in S \mid x^*$ violates $H\}$
  **if** $w(V) \leq 2w(S)/(9d-1)$
    **then for** $H \in V$ **do** $w_H \leftarrow 2w_H$ **od**;
**until** $V = \phi$;
**return** $x^*$ ;
**end** function $x^*$;
```

Figure 2: The randomized function x_i^* for LP.

3 Time complexity analysis

Two lemmas are needed for the time bound. One shows that V contains a constraint of S^*, and the other shows that V is expected to be small.

Lemma 3.1 *In the algorithms of §2, if the set V is nonempty, then it contains a constraint of S^*.*

Note that a constraint of S^* contained in V is not in V^*, since the constraints of V^* are satisfied by construction.

Proof. Suppose, in the algorithms, that x^* is a point, and that on the contrary, $V \neq \phi$ contains no constraints of S^*. Let point $x \succeq y$ if $(x_1, -\|x\|_2)$ is lexicographically greater than or equal to $(y_1, -\|y\|_2)$. We know that x^* satisfies all constraints in S^*, and so $x^*(S^*) \succeq x^*$. Since $R \cup V^* \subset S$, we know that $x^* \succeq x^*(S) = x^*(S^*)$, and so x^* has the same x_1 coordinate and norm as $x^*(S^*)$. There is only one such point in $\mathcal{P}(S^*)$, so $x^* = x^*(S^*) = x^*(S)$, and V must be empty. A similar argument holds if x^* is a ray. $\Box$

The next lemma says that V is expected to be small.

Lemma 3.2 *Let $V^* \subset S$, and let $R \subset S \setminus V^*$ be a random subset of size r, with $|S \setminus V^*| = n$. Let $V \subset S$ be the set of constraints violated by $x^*(R \cup V^*)$. Then the expected size of V is no more than $d(n - r + 1)/(r - d)$.*

Note that the slightly different meaning of n used here.

This lemma is a corollary of results in [CS89, §4]. For clarity and completeness, the proof of the results in [CS89] will be specialized for this particular case.

The intuitive idea is that since $x^*(R \cup V^*)$ violates no constraints of $R \cup V^*$, it violates few constraints of S.

After these results were first reported, Seidel found a short and elegant proof. [Sei91]

Proof. We will assume for the moment that the given problem is nondegenerate, in that no $d+1-k$ hyperplanes meet at a k-flat; in particular, no $d+1$ hyperplanes contain a common point.

We begin by putting $x^*(R \cup V^*)$ in a larger class of "candidate" optima. That is, $x^*(R \cup V^*)$ is a member of a set

$$\mathcal{F}_S = \{x^*(T \cup V^*) \mid T \subset S \setminus V^*\}.$$

We can similarly define

$$\mathcal{F}_R = \{x^*(T \cup V^*) \mid T \subset R\}.$$

Plainly $\mathcal{F}_R \subset \mathcal{F}_S$, and $x^*(R \cup V^*)$ is the unique member of $\mathcal{F}_R$ that satisfies all the constraints in R.

For a given $x \in \mathcal{F}_S$, let $|x|$ denote the number of constraints of S that it violates, and let I_x be the indicator function for x, so that $I_x = 1$ when $x = x^* = x^*(R \cup V^*)$, and $I_x = 0$ otherwise. That is, I_x is a random variable whose value depends on the random choice of R. With these definitions, the expected size of V is

$$E|V| = E \sum_{x \in \mathcal{F}_S} |x| I_x = \sum_{x \in \mathcal{F}_S} |x| E I_x = \sum_{x \in \mathcal{F}_S} |x| P_x,$$

where P_x is the probability that $x = x^*$. What is the value of P_x? Since the subsets of size r are equally likely, $P_x \binom{n}{r}$ is the number of subsets of size r that have $x = x^*$. We need to count the number of such subsets. For x to be x^*, two conditions must hold: x must be in $\mathcal{F}_R$, and x must satisfy all the constraints of R. Consider the minimal set T of constraints such that $x = x^*(T \cup V*)$. If $x \in \mathcal{F}_R$, then $T \subset R$. Moreover, the nondegeneracy assumption implies that T is unique, and has no more than d elements. Let $i_x \leq d$ denote the size of T. Then for x to be x^*, i_x given constraints must be in R, and the remaining $r - i_x$ constraints must be from among the $n - |x| - i_x$ constraints in $S \setminus V^*$ that neither define x nor are violated by x. We have

$$P_x = \binom{n - |x| - i_x}{r - i_x} \Big/ \binom{n}{r}.$$

Now since

$$\begin{aligned} \binom{n - |x| - i_x}{r - i_x} &= \frac{n - |x| - r + 1}{r - i_x} \binom{n - |x| - i_x}{r - i_x - 1} \\ &\leq \frac{n - r + 1}{r - d} \binom{n - |x| - i_x}{r - i_x - 1}, \end{aligned}$$

we have

$$E|V| \le \frac{n-r+1}{r-d} \sum_{x\in\mathcal{F}_S} |x| \binom{n-|x|-i_x}{r-i_x-1} \Big/ \binom{n}{r}.$$

The bound for $E|V|$ follows by showing that the sum is no more than d. By a counting argument as above, the summand is the probability that x is a member of $\mathcal{F}_R$ that violates exactly one constraint of R. By an indicator function argument as above, the sum is the expected number of $x \in \mathcal{F}_R$ that violate exactly one constraint of R. However, for any set R, the number of such x is no more than d: such an x is $x^*(R \cup V^* \setminus \{H\})$, for some constraint H, and $x^*(R \cup V^* \setminus \{H\}) = x^*(R \cup V^*)$ unless H is one of the d or fewer constraints determining $x^*(R \cup V^*)$.

Under the nondegeneracy assumption, we have $E|V| \le d(n-r+1)/(r-d)$. It remains to check that the complexity analysis follows through when the input set is degenerate. Given a degenerate LP problem, we will show that there is a "perturbed" version of the problem such that the expected size of the set V in the perturbed problem is no smaller than in the original. The perturbation idea goes back to [Cha52], and is equivalent to the lexicographic tie-breaking used to prevent cycling in the simplex algorithm. The idea is that the vector b is replaced by the vector $b + (\epsilon, \epsilon^2, \ldots, \epsilon^n)$, where $\epsilon > 0$ is very small. (A similar perturbation is added to the constraints for $\hat{S}$ in the unbounded case.) The resulting system is nondegenerate, in the sense discussed at the beginning of the proof. Moreover, each $x \in \mathcal{F}_S$ in the original problem is associated with a subset of $\mathcal{F}_{S'}$, where S' is the corresponding perturbed constraint set. For given $x \in \mathcal{F}_S$ and $T \subset S$ with $x = x^*(T \cup V^*)$, the optimum $x^*(T' \cup V^*)$ is in the subset associated with x, where T' is the perturbed version of T. Also, whenever $x = x^*(R \cup V^*)$, some x' associated with x is the optimum for the corresponding perturbed problem, for sufficiently small ϵ. The optimum x' violates at least as many constraints as x does. Thus the expected size of the set V in the perturbed problem is no less than that in the original problem, and the bound for $E|V|$ holds in general. □

We can use this lemma to show that progress will be made; that is, say that an execution of the loop body in x_r^* is *successful* if the test $|V| \le 2\sqrt{n}$ returns **true**, and define an analogous condition for x_i^* and x_m^*. Then the previous lemma easily implies the following.

Lemma 3.3 *The probability that any given execution of the loop body is successful is at least* $1/2$*, and so on average two executions are required to obtain a successful one.*

Proof. From the previous lemma, the expected value of $|V|$ in x_r^* is bounded by $d(n-r+1)/(r-d)$, which is no more than $\sqrt{n}$ for $r \ge d\sqrt{n}$. By Markov's inequality, the probability that $|V|$ exceeds twice its mean is no more than $1/2$.

For x_i^*, we take $V^* = \phi$ in the previous lemma, and allow S and R to be multisets with $n = w(S)$. The result follows analogously. □

Theorem 3.4 *Given an LP problem with $b \geq 0$, the iterative algorithm x_i^* requires*

$$O(d^2 n \log n) + (d \log n) O(d)^{d/2+O(1)},$$

expected time, as $n \to \infty$, where the constant factors do not depend on d.

Proof. We will show that the loop body of the algorithm is executed an expected $O(d \log n)$ times, by showing that $w(S^*)$ grows much faster than $w(S)$, so that after $O(d \log n)$ successful iterations, either $V = \phi$ or $w(S^*) > w(S)$, a contradiction.

Our argument is similar to those in [Wel88, Lit87]. By Lemma 3.1, the set V must contain a constraint of S^*. Therefore, some $H \in S^*$ is doubled in weight during a successful execution of the loop body. Let $d' = |S^*|$; after kd' successful executions of the loop body, we have $w(S^*) = \sum_{H \in S^*} w_H =$, where $w_H = 2^{i_H}$ for some i_H and $\sum_{H \in S^*} i_H \geq kd'$. It easily follows that the minimum possible value of $w(S^*)$ after kd' successful iterations is $2^k d'$.

On the other hand, when the members of V are doubled in weight, the total increase in $w(S)$ is $w(V) \leq 2w(S)/(9d-1)$. That is, upon a successful execution, the new value of $w(S)$ is no more than $(1 + 2/(9d-1))$ times the old value. After kd' successful iterations,

$$w(S) \leq (1 + 2/(9d-1))^{kd'} n \leq e^{2kd'/(9d-1)} n.$$

When $k > \ln(n/d')/(\ln 2 - 2d'/(9d-1))$, we have $w(S^*) > w(S)$, so after $kd' = O(d \log n)$ successful iterations, the set V must be empty. (In fact, r can be smaller than C_d and give the same, but no better, result.) Hence by Lemma 3.3, the iterative algorithm stops after an expected $O(d \log n)$ iterations.

It remains to bound the time required by the loop body. Vitter [Vit84] gives an algorithm for random sampling that is readily adapted to obtain weighted samples like R in $O(n)$ time, using the observation that $w(S) = n^{O(1)}$ during the algorithm. Plainly, determining V requires $O(dn)$. The simplex algorithm takes $d^{O(1)}$ time to visit a vertex of $\mathcal{P}(S)$, and visits each vertex at most once. This gives a time bound $\binom{2C_d}{\lfloor d/2 \rfloor} d^{O(1)}$ for simplex, or $O(d)^{d/2+O(1)}$ using Stirling's approximation. (This also bounds the time for finding the optimum with smallest norm.) Therefore the loop body requires $O(dn) + O(d)^{d/2+O(1)}$, and the bound follows. □

Theorem 3.5 *Algorithm x_m^* requires*

$$O(d^2 n) + (d^2 \log n) O(d)^{d/2+O(1)} + O(d^4 \sqrt{n} \log n)$$

expected time, as $n \to \infty$, where the constant factors do not depend on d.

Proof. We will continue to assume that $b > 0$ until the end of the proof.

The set V^* grows by at most $2\sqrt{n}$ at each successful iteration, with at most $d+1$ successful iterations needed. The maximum size of $R \cup V^*$ is therefore

$\sqrt{C_d n}$, where C_d is $9d^2$. Let $T_m(n)$ be the expected time required by x_m^* for a problem with n constraints (and d variables), and similarly define $T_i(n)$. Since the probability that an iteration is successful is at least $1/2$, the time required to find acceptably small sets V is bounded by $2T_i(\sqrt{C_d n})$, for a total of $2(d+1)T_i(\sqrt{C_d n})$ over all phases. The time required to test that a given constraint is satisfied by a computed optimum is $O(d)$, for a total of $O(d^2 n)$ over all phases. For $n > C_d$, the expected time $T_m(n)$ is bounded by

$$T_m(n) \leq 2(d+1)T_i(\sqrt{C_d n}) + O(d^2 n),$$

where the constant does not depend on d. The bound follows. $\square$

4 Integer linear programming

4.1 The problem

This section gives an algorithm for finding an optimum for the integer linear programming (ILP) problem

$$\max\{cx \mid Ax \leq b; x \text{ integral}\},$$

where A is an $n \times d$ matrix, b an n-vector and c a d-vector, and the entries of A, b, and c are rational numbers. As before S is the set of constraint halfspaces associated with A. In this section, $x^*(S)$ will denote the solution to the above problem (not the corresponding LP relaxation); as discussed below, we can assume that the optimum is bounded; it will be convenient here to assume that $x^*(S)$ is the lexicographically maximum point achieving the optimum value. This last condition assures that $x^*(S)$ is unique.

The optimum can be forced to be bounded by introducing new constraints. Such constraints should not change a finite optimum, however. We can use the proof of [Sch86, Cor. 17.1c] here, which shows that if φ is the facet complexity of $\mathcal{P}(S)$, and the ILP has a finite optimum, then every coordinate of that optimum has size no more than $K_d = 2d^2\varphi + \lceil \log_2(n+1) \rceil + 3$. We may add to S the set $\hat{S}$ of $2d$ constraints

$$|x_i| \leq 2^{K_d} + 1,$$

for $1 \leq i \leq d$, to obtain a bounded polytope $\mathcal{P}(S \cup \hat{S})$ with the same optimum if that optimum is finite. Moreover, no integral point in $\mathcal{P}(S \cup \hat{S})$ has size more than $7d^3\varphi$, attained when d of the new constraints are tight.

Thus we may assume that the optimum solution to the ILP is bounded; when determining $x^*(R)$ for $R \subset S$, we can make the same assumption by finding $x^*(R \cup \hat{S})$ instead. Hereafter in this section $x^*(R)$ will denote $x^*(R \cup \hat{S})$.

4.2 The algorithm

The starting point for the new algorithm for ILP is the following lemma due to Bell and to Scarf ([Bel77, Sca77]; see also [Sch86]).

Lemma 4.1 *There is a set $S^* \subset S$ with $|S^*| \leq 2^d - 1$ and with $x^*(S) = x^*(S^*)$.*

That is, as with LP, we have the optimum determined by a "small" set. The ILP algorithms are simply variations on the LP algorithms, with sample sizes using 2^d rather than d, and using Lenstra's algorithm in the base case. Another necessary modification is due to the fact that S^* is not necessarily unique. As a result, we will use the following bound, giving weaker results than Lemma 3.2. This bound is a corollary of the results in [Cla87]. (See also [HW87, Spe74].)

Lemma 4.2 *Let $V^* \subset S$, and let $R \subset S \setminus V^*$ be a random subset of size $r > 2^{d+1}$, with $|S \setminus V^*| = n$. Let $V \subset S$ be the set of constraints violated by $x^*(R \cup V^*)$. Then with probability $1/2$, $|V| \leq 2^{d+1} n (\ln r)/r$.*

Proof. We will assume that V^* is empty; the case where V^* is not empty can be handled similarly. Let $\mathcal{F}_S$ and $\mathcal{F}_R$ be defined analogously to those in Lemma 3.2. Let $m = 2^d - 1$. By the previous lemma, $x^*(R) = x^*(R')$ for some $R' \subset R$ with $|R'| \leq m$. For $k < n$, the probability that $|x^*(R)| > k$ is bounded above by

$$\sum_{0 \leq i \leq m} \sum_{\substack{R' \subset S, |R'| = i \\ |x^*(R')| > k}} \operatorname{Prob}\{x^*(R') = x^*(R)\},$$

since the probability of the union of a set of events is no more than the sum of the probabilities of those events. The probability that R contains given i-element $R' \subset S$, and that R contains none of the $|x^*(R')| > k$ constraints violated by $x^*(R')$, is no more than $\binom{n-i-k}{r-i} / \binom{n}{r}$. The number of i-element $R' \subset S$ with $|x^*(R')| > k$ is of course no more than $\binom{n}{i}$. Therefore $\operatorname{Prob}\{|x^*(R)| > k\}$ is no more than

$$\sum_{0 \leq i \leq m} \binom{n}{i} \binom{n-i-k}{r-i} \Big/ \binom{n}{r},$$

which is no more than

$$(m+1) \binom{r}{m} \binom{n-m-k}{r-m} \Big/ \binom{n-m}{r-m},$$

Using elementary bounds, this quantity is less than $1/2$ for $k \geq 2^{d+1} n (\ln r)/r$. □

The modifications for the ILP algorithms are as follows: for the recursive algorithm, put the sample size at $2^d \sqrt{2n \ln n}$, and use Lenstra's algorithm when $n \leq 2^{2d+5} d$. With probability $1/2$, the set V will have no more than

$\sqrt{2n \ln n}$ constraints; require this for a successful iteration. For the iterative algorithm, use a sample size of $2^{2d+4}(2d+4)$, with a corresponding $|V|$ bound of $n(\ln 2)/2^{d+3}$.

The analysis of the ILP mixed algorithm is similar to that for the corresponding LP algorithm: the top level does expected $2^{d+1}n$ row operations, and generates 2^{d+1} expected subproblems each with no more than $2^{d+1}\sqrt{2n \ln n}$ constraints. Solution of each subproblem requires expected $2^{d+1} \ln n$ iterations, each iteration requiring $2^{d+1}\sqrt{2n \ln n}$ row operations and a call to Lenstra's algorithm with the same number of constraints. Using the fact that the row operations are on vectors of size $O(d^3)\varphi$, and the bounds from [FT87], we have the following theorem.

Theorem 4.3 *The ILP algorithm x^*_m requires expected*

$$O(2^d n + 8^d \sqrt{n \ln n} \ln n)$$

row operations on $O(d^3\varphi)$-bit vectors, and

$$d^{O(d)}\varphi \ln n$$

expected operations on $O(d^{O(1)}\varphi)$-bit numbers, as $n \to \infty$, where the constant factors do not depend on d or φ.

5 Concluding remarks

These ideas should be applicable to other convex programming problems. For example, the problem of finding the smallest sphere enclosing a set of points in E^d is amenable to this approach, and resulting algorithm has the same time bound as the LP algorithm given. The weighted Euclidean one-center problem and various L_1 approximation problems should also be amenable.

These results may be useful in obtaining insight into the observed time complexity of various algorithms and heuristics for convex programming problems.

For $n \gg d$, the best previous result is Khachian's algorithm[Kha80], requiring $O(nd^3\varphi)$ operations on numbers with $O(d^2)\varphi$ bits. By application of Khachian's algorithm instead of simplex, an algorithm is obtained that requires $O(nd^2 \log n + d^6\varphi \log n)$ expected operations. This is an improvement when $d = o((n/\log n)^{1/3})$. (Khachian's algorithm can be adapted to the problem of finding the shortest vector in a convex set, and so minimum norm solutions can be found with it.)

It is worth noting that a variant of the iterative ILP algorithm may be useful in the (common) situation where $K = |S^*| \ll 2^d$; here a sample size $O(2^d Kd)$ will do, with fewer iterations to converge on S^*. Since K is unknown, attempts with K considered to be $d, 2d, 4d \ldots$, should be made, giving a time bound of

$$O(Kd^2 n \log n) + K^2 d^{O(d)}\varphi \log n$$

expected operations on $d^{O(1)}\varphi$-bit numbers.

Several developments have occurred since the conference version of this paper appeared. Adler and Shamir have shown that these ideas can be applied to general convex programming.[AS90] Chazelle and Matoušek have derandomized the recursive algorithm, obtaining a deterministic algorithm requiring $d^{O(d)}n$ time.[CM93] Alon and Megiddo have applied and extended the ideas of this paper to a parallel setting.[AM90]

Seidel gave a different randomized algorithm[Sei91], requiring $O(d!n)$ expected time, with a much simpler analysis; Sharir and Welzl found a variant of Seidel's algorithm requiring time subexponential in d. Their algorithm is a randomized instance of the simplex algorithm.[MSW92] Kalai was the first to find a subexponential simplex algorithm.[Kal92] Problem instances have long been known for which versions of the simplex algorithm require at least 2^d operations.[KM72] These results cast new light on the complexity of the simplex algorithm, and on the possibility that linear programming problems can be solved in "strongly polynomial" time; an algorithm with such a bound would need $(nd)^{O(1)}$ operations, with the number of operations independent of the size of the numbers specifying a problem instance.

Acknowledgements. I thank David Gay and Margaret Wright for suggesting the use of the minimum norm solution for tie-breaking. I thank Nimrod Megiddo, Rex Dwyer, Alan Frieze, and Jeff Lagarias for helpful comments.

References

[AM90] N. Alon and N. Megiddo. Parallel linear programming in fixed dimension almost surely in constant time. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, pages 574–582, 1990.

[AS90] I. Adler and R. Shamir. A randomization scheme for speeding up algorithms for linear and convex quadratic programming problems with a high constraints-to-variables ratio. Technical Report 21-90, Rutgers Univ., May 1990. To appear in *Math. Programming.*

[Bab85] L. Babai. On Lovász's lattice reduction and the nearest lattice point problem. In K. Mehlhorn, editor, *Lecture Notes in Computer Science*, pages 13–20. Springer-Verlag, 1985.

[Bel77] D. E. Bell. A theorem concerning the integer lattice. *Studies in Applied Mathematics*, 56:187–188, 1977.

[Cha52] A. Charnes. Optimality and degeneracy in linear programming. *Econometrika*, 20:160–170, 1952.

[Cla86] K. L. Clarkson. Linear programming in $O(n3^{d^2})$ time. *Information Processing Letters*, 22:21–24, 1986.

[Cla87] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete and Computational Geometry*, 2:195–222, 1987.

[CM93] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 281–290, 1993.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4:387–421, 1989.

[DF89] M. E. Dyer and A. M. Frieze. A randomized algorithm for fixed-dimensional linear programming. *Mathematical Programming*, 44:203–212, 1989.

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

[Fei84] S. D. Feit. A fast algorithm for the two-variable integer programming problem. *Journal of the ACM*, pages 99–113, 1984.

[FT87] A. Frank and É. Tardos. An application of simultaneous approximation in combinatorial optimization. *Combinatorica*, 7:49–66, 1987.

[GMW81] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, New York, 1981.

[GMW91] P. E. Gill, W. Murray, and M. H. Wright. *Numerical Linear Algebra and Optimization.* Addison-Wesley, New York, 1991.

[HW87] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Comp. Geom.*, 2:127–151, 1987.

[Kal92] G. Kalai. A subexponential randomized simplex algorithm. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 475–482, 1992.

[Kan87] R. Kannan. Minkowski's convex body theorem and integer programming. *Math. of Operations Research*, pages 415–440, 1987.

[Kha80] L. G. Khachian. Polynomial algorithms in linear programming. *Zhurnal Vychislitelnoi Mathematiki i Matematicheskoi Fiziki*, pages 53–72, 1980.

[KM72] V. Klee and G. J. Minty. How good is the simplex algorithm? In *Inequalities III*, pages 159–175. Academic Press, 1972.

[Len83] H. W. Lenstra. Integer programming with a fixed number of variables. *Math. of Operations Research*, pages 538–548, 1983.

[Lit87] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. In *Proc. 28th IEEE Symp. on Foundations of Computer Science*, pages 68–77, 1987.

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

[MSW92] 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.

[Sca77] H. E. Scarf. An observation on the stucture of production sets with indivisibilities. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 74, pages 3637–3641, 1977.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, New York, 1986.

[Sei91] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete and Computational Geometry*, pages 423–433, 1991.

[Spe74] J. Spencer. Puncture sets. *J. Combinatorial Theory A*, 17:329–336, 1974.

[Vit84] J. S. Vitter. Faster methods for random sampling. *Communications of the ACM*, pages 703–718, 1984.

[Wel88] E. Welzl. Partition trees for triangle counting and other range searching problems. In *Proc. Fourth ACM Symp. on Comp. Geometry*, pages 23–33, 1988.