A Simple Polynomial-time Rescaling Algorithm for Solving Linear Programs

John Dunagan
Microsoft Research
One Microsoft Way
Redmond, WA 98052
jdunagan@microsoft.com

Santosh Vempala
Department of Mathematics
Massachusetts Institute of Technology
Cambridge, MA 02139
vempala@math.mit.edu

Abstract

The perceptron algorithm, developed mainly in the machine learning literature, is a simple greedy method for finding a feasible solution to a linear program (alternatively, for learning a threshold function). In spite of its exponential worst-case complexity, it is often quite useful, in part due to its noise-tolerance and also its overall simplicity. In this paper, we show that a randomized version of the perceptron algorithm along with periodic rescaling runs in polynomial-time. The resulting algorithm for linear programming has an elementary description and analysis.

1 Introduction

Linear programming problems arise in many areas. The standard form is

$$\begin{aligned} \max \quad & c^T x \\ Ax \quad & \leq \quad b \\ x \quad & \geq \quad 0 \end{aligned}$$

where b, c are in $\mathbb{R}^n$, and A is an $m \times n$ matrix of reals. It is often convenient to view the constraints as halfspaces in $\mathbb{R}^n$ and the problem is to find a point in their intersection of maximum objective value. The *dual* view is to think of the rows of A as points in R^n, and then the goal is to find a threshold function (i.e., a halfspace) that satisfies the given thresholds and maximizes the objective threshold. Polynomial-time algorithms for solving linear programs include the ellipsoid method [15, 17, 27, 24], interior point methods [16, 26] and the random walk method [2].

Another classical algorithm which is often used for problems arising in machine learning is the perceptron algorithm [1, 18, 20]. It was developed to solve the problem of learning a half-space. The algorithm is a simple greedy method that is guaranteed to converge to a feasible solution, if one exists. It could take an exponential number of iterations in the worst case. Nevertheless, it has many useful properties, including certain types of noise tolerance [4, 5]. It has also been related to boosting in the PAC model of learning [22, 23]. It was recently shown that the algorithm is polynomial with high probability for randomly perturbed linear programs [3]. It should be noted that the algorithm is generally not considered efficient in practice. It has been an open question as to whether there is a variant of the perceptron algorithm that is polynomial in the worst case. Besides suggesting a useful modification to the basic algorithm in practice, it would give a noise-tolerant (in a specific sense) polynomial-time algorithm.

We focus on the problem of finding a feasible solution to a given set of linear constraints. Polynomial time reductions of the optimization problem to the feasibility version of the problem are well-known [15]. A typical approach is described in Section 4.

In this paper, we show that a randomized perceptron-like algorithm along with periodic rescaling applied to a feasible linear program will return a feasible point in polynomial time, with high probability. It takes as input an arbitrary set of linear constraints and an additional parameter δ and outputs the correct answer with probability at least $1 - \delta$. As in all variants of the perceptron algorithm, it does not use any matrix inversions or barrier functions. Our main theorem (Theorem 3.5) is that a strictly feasible linear program with m constraints in $\mathbb{R}^n$ is solved in time $O(mn^4 \log n \log(1/\rho_0) + mn^3 \log n \log(1/\delta))$ where ρ_0 is a parameter that roughly corresponds to the radius of a ball that fits in the feasible region, and $\log(1/\rho_0)$ is guaranteed to be bounded by a polynomial in the input description. It should be noted that the complexity of our algorithm is not as good as the current best algorithms [10, 26]. On the other hand, it is rather simple and inherits the noise-tolerance properties of the perceptron algorithm. As a direct consequence, we get a simpler (and faster) solution to the problem of learning noisy linear threshold functions [4, 6].

Here is the main idea. The perceptron algorithm maintains a halfspace that it updates using a constraint that is currently violated. The convergence of the algorithm depends on a separation parameter quantifying the amount of "wiggle room" available for a feasible halfspace. This is in the dual view described above. In the corresponding primal view, when we think of constraints as halfspaces, the wiggle room is the radius of the largest ball that fits in the feasible region. Roughly speaking, the analysis of the perceptron algorithm says that if this ball is large, then the algorithm will converge quickly. The work of [4] shows that even when this ball is small, a simple variant of the perceptron algorithm finds a *nearly* feasible solution, i.e., constraints might be violated but each is not violated by much. Here we show that when the wiggle room is small, such a nearly feasible solution can be used to apply a linear transformation (in fact, a rank 1 update) that expands the wiggle room (enlarges the ball contained in the feasible region) by roughly a $1 + 1/n$ factor. Thus, in $O(n)$ iterations, we either find a feasible point or double the wiggle room. The idea of scaling to improve convergence appears in the work of Shor [25].

2 The Algorithm

In this section we present an algorithm for the linear feasibility problem

$$Ax \geq 0, x \neq 0$$

consisting of m constraints in n dimensions, i.e. $x \in \mathbb{R}^n$ and A is $m \times n$. In Section 4 we show how well-known methods can be used to reduce the standard form to this *homogenized* form.

The algorithm is iterative and each iteration consists of three phases, a *perceptron* phase, a *perceptron improvement* phase, and a *rescaling* phase. The perceptron phase uses the classical perceptron algorithm. The perceptron improvement phase uses a modified version of the basic perceptron algorithm. This modified version was described earlier in [4]. Figure 1 depicts the rescaling phase.

The classical perceptron algorithm for the linear feasibility problem is the following: find a violated constraint, move the trial solution x one unit in the direction normal to the violated constraint, and repeat if necessary (this is step 2 in the algorithm).

In the rest of the paper, we let $\bar{x}$ denote the unit vector in the direction of x.

3 Analysis

3.1 Ideas

Let A_0 denote the initial matrix input to the algorithm, and let A_* denote the matrix that the algorithm terminates with. When the algorithm terminates, it produces a non-zero vector Bx such that $A_*x = (A_0B)x \geq 0$, i.e., $A_0(Bx) \geq 0$, as desired.

During each outer iteration (Steps 2-7), the perceptron improvement phase (Step 4) may be run a number of times, but it terminates quickly with high probability (Lemma 3.2). Thus the main question is, how many iterations does the algorithm take? To answer this, we use the following quantity which measures the "roundness" of the feasible region (introduced in [13, 14] as the *inner measure*):

$$\rho(A) = \max_{x:||x||=1, Ax\geq 0} \min_i (\bar{a}_i \cdot x)$$

where a_i denotes the i'th row of A and $\bar{a}_i$ is the unit vector along a_i.

```
Algorithm.

Input: An $m \times n$ matrix $A$.
Output: A point $x$ such that $Ax \geq 0$ and $x \neq 0$.

  1. Let $B = I, \sigma = 1/(32n)$.
  2. (Perceptron)
      (a) Let $x$ be the origin in $\mathbb{R}^n$.
      (b) Repeat at most $1/\sigma^2$ times:
          If there exists a row $a$ such that $a \cdot x \leq 0$, set $x = x + \bar{a}$.
  3. If $Ax \geq 0$, then output $Bx$ as a feasible solution and stop.
  4. (Perceptron Improvement)
      (a) Let $x$ be a uniform random unit vector in $\mathbb{R}^n$.
      (b) Repeat at most $(\ln n)/\sigma^2$ times:
          If there exists a row $a$ such that $\bar{a} \cdot \bar{x} < -\sigma$, set $x = x - (\bar{a} \cdot x)\bar{a}$.
          If $x = 0$, go back to step (a).
      (c) If there is still a row $a$ such that $\bar{a} \cdot \bar{x} < -\sigma$, restart at step (a).
  5. If $Ax \geq 0$, then output $Bx$ as a feasible solution and stop.
  6. (Rescaling)
          Set $A = A\left(I + \bar{x}\bar{x}^T\right)$   and   $B = B\left(I + \bar{x}\bar{x}^T\right)$.
  7. Go back to step 2.
```

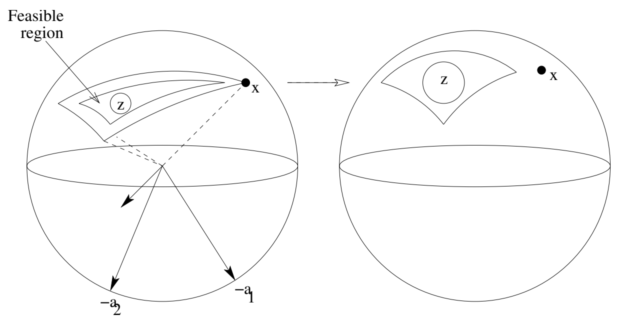

Figure 1: A constraint system before and after rescaling.

We call ρ the *radius* of A, and z, the unit vector that achieves the maximum, its *center* (this might not be unique and any choice among the points achieving the maximum will do). Note that ρ is just the radius of the largest ball that fits in the feasible cone, such that the center of the ball is on the unit sphere. To avoid confusion, let $\rho_0 = \rho(A_0)$ denote the roundness of the input. See Section 4 for a further discussion of ρ.

The classical analysis of the perceptron algorithm [18] (repeated in Lemma 3.1) shows that the classical perceptron algorithm (our perceptron phase) applied to a linear feasibility problem with radius ρ will terminate in at most $1/\rho^2$ iterations. Further, it is not hard to see that if ρ is initially more than 0, there exists a linear transformation that takes ρ arbitrarily close to 1. This transformation is defined by $A' = A(I + \nu zz^T)$. As $\nu \to \infty$, a simple calculation shows that $\rho(A')$ goes to 1. So if we knew the transformation, we could just apply it once and then run the standard perceptron algorithm!

However, this is equivalent to the original problem since z is a feasible point. Instead, in our algorithm, we incrementally transform A using *near-feasible* solutions, so that ρ increases steadily. Our main lemma shows that in any iteration of our algorithm where ρ is small, it will increase in expectation by a multiplicative factor. Combining our main lemma with the classical analysis will yield that if ρ is small, it gets bigger (guaranteed by the perceptron improvement and scaling phases), and if ρ is big, the algorithm finds a feasible point (guaranteed by the perceptron phase).

Each iteration of the algorithm consists a perceptron phase and a perceptron improvement phase. Alternatively, one could do just the perceptron improvement phase for a pre-determined number of steps, and then check for completion by running the perceptron phase once.

3.2 Proofs

We begin with the well-known analysis of the standard perceptron algorithm.

Lemma 3.1 (Block-Novikoff [18]) *The classical perceptron algorithm (our perceptron phase) returns a feasible point in at most $1/\rho^2$ iterations.*

Proof. Consider the potential function $x \cdot z / \|x\|$. The numerator increases by at least ρ on each step:

$$(x + \bar{a}_i) \cdot z = x \cdot z + \bar{a}_i \cdot z \geq x \cdot z + \rho$$

While the square of the denominator increases by at most 1:

$$(x + \bar{a}_i) \cdot (x + \bar{a}_i) = x \cdot x + 2x \cdot \bar{a}_i + \bar{a}_i \cdot \bar{a}_i \leq x \cdot x + 1$$

since $x \cdot \bar{a}_i \leq 0$. After t iterations, the potential function is at least $\frac{t\rho}{\sqrt{t}}$ and thus the classical perceptron algorithm must terminate before $1/\rho^2$ iterations. If the algorithm terminates, it must have found a feasible point. $\square$

Next, we recall the analysis of the modified perceptron algorithm (our perceptron improvement phase).

Lemma 3.2 (BFKV [4]) *Let A be the constraint matrix at the beginning of a perceptron improvement phase and let z be any unit vector such that $Az \geq 0$. With probability at least $1/8$, in at most*

$(\ln n)/\sigma^2$ *steps, the perceptron improvement phase returns a vector x such that both conditions below hold:*

$$\begin{array}{lll} (a) & \bar{a} \cdot x \geq -\sigma & \text{for every row } a \text{ of } A \\ (b) & z \cdot \bar{x} \geq \dfrac{1}{\sqrt{n}}. & \end{array}$$

Proof. The proof of both parts is similar. A standard computation shows that for a random unit vector x, $z \cdot \bar{x} \geq 1/\sqrt{n}$ with probability at least $1/8$ (for a proof outline of this folklore fact, see the appendix). We now show that if this is the case, we terminate in the desired number of iterations.

Note that in each update step, $z \cdot x$ does not decrease

$$(x - (x \cdot \bar{a}_i)\bar{a}_i) \cdot z = x \cdot z - (x \cdot \bar{a}_i)(\bar{a}_i \cdot z) \geq x \cdot z$$

because $x \cdot \bar{a}_i$ had to be negative in order for $\bar{a}_i$ to be used in an update step, and $\bar{a}_i \cdot z \geq 0$ by assumption. (This also implies that if $z \cdot x \geq 1/\sqrt{n}$ initially, x will never be set to zero.) On the other hand $x \cdot x$ does decrease significantly because

$$\begin{aligned} (x - (x \cdot \bar{a}_i)\bar{a}_i) \cdot (x - (x \cdot \bar{a}_i)\bar{a}_i) &= x \cdot x - 2(\bar{a}_i \cdot x)^2 + (\bar{a}_i \cdot x)^2 \\ &= x \cdot x - (\bar{a}_i \cdot x)^2 \\ &\leq x \cdot x(1 - \sigma^2). \end{aligned}$$

Thus after t iterations $\|x\| \leq (1 - \sigma^2)^{t/2}$. If $t > (\ln n)/\sigma^2$, we would have $\frac{x \cdot z}{\|x\|} > 1$, which cannot happen. Therefore, every time we start through this phase, with probability at least $1/8$ we terminate and return a vector x for which

$$\frac{x \cdot z}{\|x\|} \geq \frac{1}{\sqrt{n}}.$$

□

We are now ready to prove our main lemma about progress in any iteration that does not find a feasible solution.

Lemma 3.3 *Suppose that $\rho, \sigma \leq 1/(32n)$. Let A' be obtained from A by one iteration of the algorithm (one on which the problem was not solved). Let ρ' and ρ be the radii of A' and A respectively. Then,*

(a) $\rho' \geq (1 - \frac{1}{32n} - \frac{1}{512n^2})\rho$.

(b) With probability at least $\frac{1}{8}$, $\rho' \geq (1 + \frac{1}{3n})\rho$.

Proof. The analysis will use Lemma 3.2 which says that, with probability $\frac{1}{8}$, the vector x at the end of step 4(b) satisfies $\bar{a} \cdot \bar{x} \geq -\sigma$ for every constraint row a and $z \cdot \bar{x} \geq 1/\sqrt{n}$.

Let a_i, $i = 1, \ldots m$ be the rows of A at the beginning of some iteration. Let z be a unit vector satisfying $\rho = \min_i \bar{a}_i \cdot z$, and let $\sigma_i = \bar{a}_i \cdot \bar{x}$. After a perceptron improvement phase, we get a vector x such that for all i,

$$\bar{a}_i \cdot \bar{x} = \sigma_i \geq -\sigma.$$

As in the theorem statement, let A' be the matrix obtained after the rescaling step, i.e.

$$a_i' = a_i + (a_i \cdot \bar{x})\bar{x}.$$

Finally, define

$$z' = z + \beta\bar{x}.$$

where β will be specified shortly. Although z' is not necessarily the center of A', ρ' is a maximum over a set, and so considering one element ($\bar{z}'$) of the set suffices to lower bound ρ'. We have

$$\rho' \geq \min_i \bar{a}_i' \cdot \bar{z}' = \min_i \frac{\bar{a}_i' \cdot z'}{||z'||}.$$

We will first prove that $\bar{a}_i' \cdot z'$ cannot be too small.

$$\begin{aligned} \bar{a}_i' \cdot z' &= \left(\frac{\bar{a}_i + (\bar{a}_i \cdot \bar{x})\bar{x}}{\|\bar{a}_i + (\bar{a}_i \cdot \bar{x})\bar{x}\|}\right) \cdot z' \\ &= \frac{(\bar{a}_i + (\bar{a}_i \cdot \bar{x})\bar{x})(z + \beta\bar{x})}{\sqrt{1 + 3(\bar{a}_i \cdot \bar{x})^2}} \\ &\geq \frac{\rho + (\bar{a}_i \cdot \bar{x})(z \cdot \bar{x}) + 2\beta(\bar{a}_i \cdot \bar{x})}{\sqrt{1 + 3\sigma_i^2}} \end{aligned}$$

We choose:

$$\beta = \frac{1}{2}(\rho - (\bar{x} \cdot z)).$$

We proceed to calculate

$$\bar{a}_i' \cdot z' \geq \rho \frac{1 + \sigma_i}{\sqrt{1 + 3\sigma_i^2}} \geq \rho \frac{1 - \sigma}{\sqrt{1 + 3\sigma^2}}. \tag{1}$$

where the second inequality follows from $\sigma_i \in [-\sigma, 1]$. Next, observe that

$$||z'||^2 = 1 + \beta^2 + 2\beta(\bar{x} \cdot z) = 1 + \frac{\rho^2}{4} + (z \cdot \bar{x})\left(\frac{\rho}{2} - \frac{3}{4}(z \cdot \bar{x})\right).$$

We consider two cases:

1. $|z \cdot \bar{x}| < \frac{1}{\sqrt{n}}$. This happens with probability at most 7/8.

   Viewing $||z'||^2$ as a quadratic polynomial in $(z' \cdot x)$, we see that it is maximized when $(z' \cdot x) = \frac{\rho}{3}$. In this case, we have

   $$||z'||^2 \leq 1 + \frac{\rho^2}{4} + \frac{\rho^2}{12} = 1 + \frac{\rho^2}{3}.$$

   Using the elementary inequality $\frac{1}{\sqrt{1+\beta}} \geq 1 - \frac{\beta}{2}$ for $\beta \in (-1, 1)$, we find

   $$\begin{aligned} \rho' &\geq \rho \frac{1 - \sigma}{\sqrt{1 + 3\sigma^2}||z'||} \\ &\geq \rho(1 - \sigma)(1 - \frac{3\sigma^2}{2})(1 - \frac{\rho^2}{6}) \\ &\geq \rho(1 - \sigma - \frac{3\sigma^2}{2} - \frac{\rho^2}{6}) \\ &\geq \rho(1 - \frac{1}{32n} - \frac{1}{512n^2}) \end{aligned}$$

since $\sigma, \rho \leq 1/(32n)$.

2. $|z \cdot \bar{x}| \geq \frac{1}{\sqrt{n}}$. This happens with probability at least 1/8.

   In this case,

$$||z'||^2 = 1 + \frac{\rho^2}{4} + (z \cdot \bar{x})\frac{\rho}{2} - \frac{3}{4}(z \cdot \bar{x})^2 \leq 1 - \frac{3}{4n} + \frac{\rho}{2\sqrt{n}} + \frac{\rho^2}{4}.$$

   Using the elementary inequality again, we find

$$\begin{aligned} \rho' &\geq \rho(1-\sigma)(1 - \frac{3\sigma^2}{2})(1 + \frac{3}{8n} - \frac{\rho}{4\sqrt{n}} - \frac{\rho^2}{8}) \\ &\geq \rho\left(1 - \sigma - \frac{3\sigma^2}{2} + \frac{3}{8n} - \frac{3\sigma}{8n} - \frac{9\sigma^2}{16n} - \frac{\rho}{4\sqrt{n}} - \frac{\rho^2}{8}\right) \\ &\geq \rho\left(1 + \frac{1}{3n}\right). \end{aligned}$$

This proves both parts of the lemma. □

We can now bound the number of iterations. Recall that $\rho_0 = \rho(A_0)$ is the ρ value for the initial input matrix.

Theorem 3.4 *For any $\delta \in [0, 1]$, with probability at least $1 - \min\{\delta, e^{-n}\}$, the algorithm finds a feasible solution in $O(\log(1/\delta) + n \log(1/\rho_0))$ iterations.*

Proof. It suffices to show that ρ will be at least $1/(32n)$ in $O(\log(1/\delta) + n \log(1/\rho_0))$ iterations with probability at least $1 - \min\{\delta, e^{-n}\}$. Let X_i be a random variable for the i'th iteration, with value 1 if ρ grows by a factor of $(1 + 1/3n)$ or more and value 0 otherwise. Strictly speaking, these random variables are not i.i.d.. So, we consider another sequence, Y_i, such that $Y_i = 1$ if $|z \cdot \bar{x}| \geq 1/\sqrt{n}$ and $Y_i = 0$ otherwise. Then the Y_i are i.i.d. and by Lemma 3.3(b), $X_i \geq Y_i$. Let $X = \sum_{i=1}^{T} X_i$ and $Y = \sum_{i=1}^{T} Y_i$. Then $X \geq Y$,

$$\mathbf{E}[Y] = T \Pr(Y_1 = 1) \geq \frac{T}{8},$$

and the Chernoff bound gives

$$\Pr(Y < (1-\epsilon)\mathbf{E}[Y]) \leq e^{-\epsilon^2 \mathbf{E}[Y]/2} = e^{-\epsilon^2 T/16}$$

Let ρ_T be the ρ value after T iterations. Let $T = 4096 \max\{n \ln(1/\rho_0), \ln(1/\delta)\}$ and $\epsilon = 1/16$. Then, using the fact that $\rho_0 < 1/(32n)$,

$$e^{-\epsilon^2 T/16} < \min\{\delta, e^{-n}\}.$$

Analyzing ρ_T in the case that Y is within ϵ of its expectation, we have

$$\begin{aligned}
\rho_T &\geq \rho_0 \left(1+\frac{1}{3n}\right)^X \left(1-\frac{1}{32n}-\frac{1}{512n^2}\right)^{T-X} \\
&\geq \rho_0 \left(1+\frac{1}{3n}\right)^Y \left(1-\frac{1}{32n}-\frac{1}{512n^2}\right)^{T-Y} \\
&\geq \rho_0 \left(1+\frac{1}{3n}\right)^{\frac{T}{8}-\epsilon\frac{T}{8}} \left(1-\frac{1}{32n}-\frac{1}{512n^2}\right)^{\frac{7T}{8}+\epsilon\frac{T}{8}} \\
&\geq \rho_0 \left(1+\frac{1}{3n}\right)^{\frac{15T}{128}} \left(1-\frac{1}{32n}-\frac{1}{512n^2}\right)^{\frac{113T}{128}} \\
&\geq \rho_0 e^{T/1000n}.
\end{aligned}$$

In summary, with probability at least $1-\min\{\delta, e^{-n}\}$, in at most T iterations, ρ grows to at least $1/(32n)$, at which point the perceptron phase succeeds in finding a feasible point. $\square$

Finally, the time complexity is a matter of accounting.

Theorem 3.5 *For any $\delta \in [0,1]$, with probability at least $1-\min\{\delta, e^{-n}\}$, the algorithm finds a feasible solution in time $O(mn^4 \log n \log(1/\rho_0) + mn^3 \log n \log(1/\delta))$.*

Proof. The inner loops of the perceptron phase and the perceptron improvement phase require at most one matrix-vector multiplication (time $O(mn)$), and a constant number of vector manipulations (time $O(n)$). The number of times we repeat the inner loop is $O(n^2)$ in the perceptron phase (Step 2(b)), and at most $\log n/\sigma^2 = O(n^2 \log n)$ in the perceptron improvement phase (Step 4(b)). The scaling phase takes time $O(mn)$. Computing Bx takes time $O(n^2)$. This gives a total of $O(mn^3 \log n)$ per iteration.

In the bound on the number of iterations in the previous theorem, every pass through Step 4(b) is counted as one iteration. Using the bound of $O(n\log(1/\rho)+\log(1/\delta))$ on the number iterations, we get the overall time bound as claimed. $\square$

4 The Standard Form and Polynomiality

In this section, we discuss how to reduce the standard linear programming problem to the one solved in this paper and conclude with a discussion of the complexity bound. A typical approach to reduce optimization to feasiblity is to replace max $c^T x$ by the constraint $c^T x \geq c_0$ for some value of c_0. Binary search can be used to determine a nearly optimal value of c_0. A solution that is feasible for the problem with c_0 sufficiently close to optimal can be *rounded* to an optimal vertex solution by moving to a basic feasible solution [15].

Next, we show how to reduce the standard form linear feasibility problem

$$Ax \leq b, x \geq 0$$

into the linear feasibility problem we studied earlier. This technique is typically referred to as *homogenization.*

Introduce the variable x_0 and consider the problem

$$Ax \leq bx_0, x \geq 0, x_0 > 0$$

To convert a solution for the standard form to one for the homogenized form, set $x_0 = 1$. To convert a solution from the homogenized form to a solution for the standard form, divide x by x_0. To rewrite the homogenized form as just

$$A'x' \geq 0, x' \neq 0,$$

let $x' = [x \ x_0]$ and

$$A' = \begin{bmatrix} -A \ b \\ I \ 0 \\ 0 \ 1 \end{bmatrix}$$

A valid solution to $A'x' \geq 0$ might have $x_0 = 0$. However, because the classic perceptron algorithm (our perceptron phase) always produces solutions in the strict interior of the feasible region, our algorithm will always return a solution with $x_0 > 0$.

We now discuss the complexity of the algorithm. The traditional measure of the difficulty of a linear programming problem in the Turing model of computation is the "bit-length" denoted by L. The quantity L is essentially the input length of the linear program. Since the total number of operations in the algorithm is polynomial, it suffices to maintain a polynomial number of bits of accuracy for all our computations (the intermediate results can be irrational since we compute square roots). This is similar to many other linear programming algorithms. Another issue is that our stated complexity depends on $\log(1/\rho_0)$ where ρ_0 is not explicitly part of the input. In particular, ρ_0 for a given input might be zero, e.g., if the feasible region is not full-dimensional. This issue is also common to linear programming algorithms and can be resolved as follows. Suppose $P = \{Ax \leq b\}$ is the given linear program. Then we consider

$$P' = \{Ax \leq b + \bar{\epsilon}\}$$

where $\bar{\epsilon}$ is an all-ϵ column vector and ϵ is chosen small enough so that P is feasible iff P' is feasible. It is well-known (see e.g. [21]) that $\log(1/\epsilon)$ can be bounded by a polynomial in the input length. The set P' is full-dimensional and moreover $\log(1/\rho_0(P'))$ is bounded by a polynomial in the input length. Alternatively, one can use condition numbers and methods to bound them [11, 7]. It was shown in [9] that ρ of the homogenized program is no more than n times the Renegar condition number [19] of the original program, which in turn can be bounded by applying a small perturbation.

Finally, we note that our algorithm is randomized and it can fail with probability at most any desired δ with the complexity growing as $\log(1/\delta)$. For a feasible LP, the algorithm will find a feasible solution in the prescribed time bound with probability $1 - \delta$ (in fact, the failure probability is at most $\min\{\delta, e^{-n}\}$). Thus, if the algorithm fails to find a feasible solution, one can conclude that the input LP is infeasible and this is guaranteed to be correct with probability at least $1 - \delta$. In other words, for an infeasible LP the algorithm always concludes that it is infeasible, while for a feasible LP it finds a solution with probability $1 - \delta$ and (incorrectly) concludes that it is infeasible with probability at most δ.

5 Acknowledgements

We thank Dan Stefankovic, Adam Smith for useful discussions, and Adam Klivans and Rocco Servedio for bringing [12] to our attention. We are grateful to Rob Freund and Mario Szegedy for numerous useful comments. We thank the anonymous referees for many improvements.

References

[1] S. Agmon, The relaxation method for linear inequalities, *Canadian J. of Math.*, 6(3), 382–392, 1954.

[2] D. Bertsimas and S. Vempala, Solving convex programs by random walks, *Journal of the ACM* 51(4), 540–556, 2004.

[3] A. Blum and J. Dunagan, Smoothed Analysis of the Perceptron Algorithm for Linear Programming, *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 905–914, 2002.

[4] A. Blum, A. Frieze, R. Kannan, and S. Vempala, A polynomial-time algorithm for learning noisy linear threshold functions, *Algorithmica*, 22(1), 35–52, 1998.

[5] T. Bylander, Learning linear threshold functions in the presence of classification noise, *Proceedings of the Workshop on Computational Learning Theory*, 1994.

[6] E. Cohen, Learning noisy perceptrons by a perceptron in polynomial time, *Proceedings of the Annual IEEE Symposium on the Foundations of Computer Science*, 514–523, 1997.

[7] F. Cucker and D. Cheung, A new condition number for linear programming, *Mathematical Programming, Series A*, 91(1), 163–174, 2001.

[8] V. Chvatal, *Linear Programming*, W.H. Freeman, 1983.

[9] J. Dunagan, S. Teng, and D. A. Spielman, Smoothed Analysis of Renegar's Condition Number for Linear Programming, *SIAM Conference on Optimization*, 2002.

[10] R. M. Freund and S. Mizuno: "Interior Point Methods: Current Status and Future Directions," in High Performance Optimization, H. Frenk et al. (eds.), Kluwer Academic Publishers, pp. 441-466, 2000.

[11] R. M. Freund and J. R. Vera, Some characterizations and properties of the "distance to ill-posedness" and the condition measure of a conic linear system, *Math. Programming*, 86, 225–260, 1999.

[12] J. Forster, A Linear Lower Bound on the Unbounded Error Probabilistic Communication Complexity, *Sixteenth Annual IEEE Conference on Computational Complexity 2001*, http://citeseer.nj.nec.com/forster01linear.html

[13] J.-L. Goffin, On the Finite Convergence of the Relaxation Method for Solving Systems of Inequalities, *Ph.D Thesis*, 1971. Also research report of the Operations Research Center, University of California at Berkeley.

[14] J.-L. Goffin, The relaxation method for solving systems of linear inequalities, *Math. of Oper. Res.* 5(3), 388–414, 1980.

[15] L. Grötchel, L. Lovasz, and A. Schrijver, *Geometric algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.

[16] N. Karmarkar, A new polynomial-time algorithm for linear programming, *Combinatorica*, 4, 373–396, 1984.

[17] L. G. Khachiyan, A polynomial algorithm in linear programming, (in Russian), *Doklady Akedamii Nauk SSSR*, 244, 1093–1096, 1979 (English translation: *Soviet Mathematics Doklady*, 20, 191–194, 1979).

[18] M. L. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry*, 1969.

[19] J. Renegar, Incorporating condition measures into the complexity theory of linear programming, *SIAM Journal on Optimization*, 5(3), 506–524, 1995.

[20] F. Rosenblatt, *Principles of Neurodynamics*, Spartan Books, 1962.

[21] A. Schrijver, *Theory of Linar and Integer Programming*, Wiley, 1998.

[22] R. Servedio. On PAC Learning using Perceptron, Winnow and a Perceptron-Like Algorithm. *SIAM Journal on Computing*, 31(5), 1358-1369, 2002.

[23] R. Servedio, Smooth Boosting and Learning with Malicious Noise. Fourteenth Annual Conference on Computational Learning Theory, 473-489, 2001.

[24] N. Z. Shor, Cut-off method with space extension in convex programming problems, *Cybernetics*, 13, 94–96, 1977.

[25] N. Z. Shor, Utilization of the operation of space dilation in the minimization of convex functions, *Kibernetika*, 1, 6–12, 1970. English translation: *Cybernetics*, 1, 7–15.

[26] P. M. Vaidya, A new algorithm for minimizing convex functions over convex sets, *Mathematical Programming*, 291–341, 1996.

[27] D. B. Yudin and A. S. Nemirovski, Evaluation of the information complexity of mathematical programming problems, (in Russian), Ekonomika i Matematicheskie Metody 12, 128-142, 1976 (English translation: Matekon 13, 2, 3-45, 1976).

6 Appendix

Here we will show that for a fixed unit vector $z \in \mathbb{R}^n$, the probability that a random unit vector $x \in \mathbb{R}^n$ satisfies $z \cdot x \geq 1/\sqrt{n}$ is at least 1/8. Throughout, we assume $n \geq 4$.

We begin with a proof of a constant lower bound on the probability. Let S_{n-1} be the unit sphere in $\mathbb{R}^n$ and $C(t)$ denote the cap at distance t from the origin along the x_1 axis, i.e.,

$$C(t) = \{y \in S_{n-1} : y_1 \geq t\}$$

The probability of the desired event is $\mathrm{vol}(C(1/\sqrt{n}))/\mathrm{vol}(S_{n-1})$.

Expressing these volumes as integrals in terms of an angle θ, we get

$$\frac{\int_{\arcsin(1/\sqrt{n})}^{\pi/2} (\cos\theta)^{n-2}\ d\theta}{2\int_0^1 (\cos\theta)^{n-2}\ d\theta}.$$

Using the substitution $t = \sin\theta$, this is equal to

$$\frac{\int_{1/\sqrt{n}}^{1} (1-t^2)^{(n-3)/2}\, dt}{2\int_0^1 (1-t^2)^{(n-3)/2}\, dt}.$$

The integrand $A(t) = (1-t^2)^{(n-3)/2}$ has a maximum value of 1, so its integral in the range $[0, 1/\sqrt{n}]$ is at most $1/\sqrt{n}$. On the other hand, in the range $[1/\sqrt{n}, 2/\sqrt{n}]$, assuming $n \geq 5$, the integrand is at least

$$\left(1 - \frac{4}{n}\right)^{(n-3)/2} \geq \frac{1}{e^2}$$

and so the integral in the range $[1/\sqrt{n}, 1]$ is at least $1/(e^2\sqrt{n})$. Hence the desired event has probability at least

$$\frac{\int_{1/\sqrt{n}}^{1} A(t)\, dt}{2\int_0^{1/\sqrt{n}} A(t)\, dt + 2\int_{1/\sqrt{n}}^{1} A(t)\, dt} \geq \frac{\frac{1}{e^2\sqrt{n}}}{2\left(\frac{1}{\sqrt{n}} + \frac{1}{e^2\sqrt{n}}\right)} = \frac{1}{2(1+e^2)},$$

an absolute constant.

To calculate the bound more precisely, it is useful to view a random unit vector being generated as follows: we pick each coordinate independently from a standard Gaussian and then scale the resulting vector to make it unit length. Let the random variables representing the coordinates be $X_1, \ldots, X_n$. Then we are interested in the following:

$$\Pr\left(\frac{X_1}{\sqrt{\sum_{i=1}^n X_i^2}} \geq \frac{1}{\sqrt{n}}\right) = \frac{1}{2}\Pr\left(X_1^2 \geq \frac{\sum_{i=2}^n X_i^2}{n-1}\right).$$

Each X_i^2 has a χ-squared distribution. At this point we could use a concentration bound on the sum of $n-1$ such variables. Alternatively, one can first observe that the desired probability is a monotonic decreasing function of n and then calculate its limit as n increases. In the limit, the desired probability is

$$\frac{1}{2}\Pr(X_1^2 \geq 1) = \Pr(X_1 \geq 1)$$

where X_1 is drawn from the standard Normal distribution. This quantity is well-studied and is a constant slightly larger than 1/8.