18
The Centroid and Ellipsoid Algorithms

In this chapter, we discuss some algorithms for convex programming that have $O(\log 1/\varepsilon)$-type convergence guarantees (under suitable assumptions). This leads to polynomial-time algorithms for Linear Programming problems. In particular, we examine the Center-of-Gravity and Ellipsoid algorithms in depth.

18.1 The Centroid Algorithm

In this section, we discuss the Centroid Algorithm in the context of constrained convex minimization. Besides being interesting in its own right, it is a good lead-in to Ellipsoid, since it gives some intuition about high-dimensional bodies and their volumes.

Given a convex body $K \subseteq \mathbb{R}^n$ and a convex function $f : \mathbb{R}^n \to \mathbb{R}$, we want to approximately minimize $f(x)$ over $x \in K$. First, recall that the ***centroid*** of a set K is the point $c \in \mathbb{R}^n$ such that

$$c := \frac{\int_{x \in K} x \, dx}{\text{vol}(K)} = \frac{\int_{x \in K} x \, dx}{\int_{x \in K} dx},$$

This is the natural analog of the centroid of n points $x_1, x_2, \ldots, x_N$, which is defined as $\frac{\sum_i x_i}{N}$. See where? this blog post for a discussion about the centroid of an arbitrary measure μ defined over $\mathbb{R}^n$.

where $\text{vol}(K)$ is the volume of the set K. The following lemma captures the crucial fact about the center-of-gravity that we use in our algorithm.

Lemma 18.1 (Grünbaum's Lemma)**.** *For any convex set $K \in \mathbb{R}^n$ with a centroid $c \in \mathbb{R}^n$, and any halfspace $H = \{x \mid a^\intercal(x - c) \geq 0\}$ passing through c,*

$$\frac{1}{e} \leq \frac{\text{vol}(K \cap H)}{\text{vol}(K)} \leq \left(1 - \frac{1}{e}\right).$$

This bound is the best possible: e.g., consider the probability simplex Δ_n with centroid $\frac{1}{n}\mathbb{1}$. Finish this argument.

18.1.1 The Algorithm

In 1965, A. Yu. Levin and Donald Newman independently (and on

Levin (1965)
Newman (1965)

opposite sides of the iron curtain) proposed the following algorithm.

Algorithm 17: Centroid(K, f, T)

```
K_1 ← K
for t = 1, ... T do
    at step t, let c_t ← centroid of K_t
    K_{t+1} ← K_t ∩ {x | ⟨∇f(c_t), x − c_t⟩ ≤ 0}
return x̂ ← arg min_{t∈{1,...,T}} f(c_t)
```

The figure to the right shows a sample execution of the algorithm, where K is initially a ball. (Ignore the body K^ε for now.) We find the centroid c_1 and compute the gradient $\nabla f(c_1)$. Instead of moving in the direction opposite to the gradient, we consider the halfspace H_1 of vectors negatively correlated with the gradient, restrict our search to $K \leftarrow K \cap H_1$, and continue. We repeat this step some number of times, and then return the smallest of the function value at all the centroids seen by the algorithm.

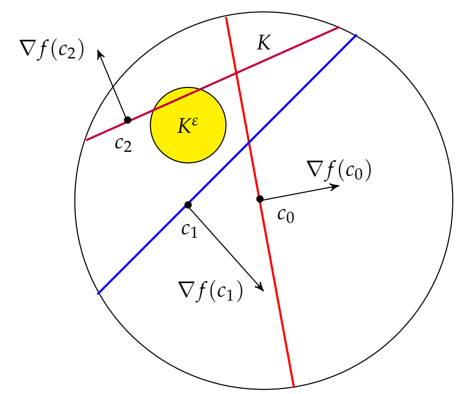

Figure 18.1: Sample execution of first three steps of the Centroid Algorithm.

18.1.2 An Analysis of the Centroid Algorithm

Theorem 18.2. *Let $B \geq 0$ such that $f : K \to [-B, B]$. If $\widehat{x}$ is the result of the algorithm, and $x^* = \arg\min_{x \in K} f(x)$, then*

$$f(\widehat{x}) - f(x^*) \leq 4B \cdot \exp(-T/3n).$$

Hence, for any $\varepsilon \leq 1$, as long as $T \geq 3n \ln \frac{4B}{\varepsilon}$,

$$f(\widehat{x}) - f(x^*) \leq \varepsilon.$$

Proof. For some $\delta \leq 1$, define the body

$$K^\delta := \{(1-\delta)x^* + \delta x \mid x \in K\}$$

as a scaled-down version of K centered at x^*. The following facts are immediate:

1. $\text{vol}(K^\delta) = \delta^n \cdot \text{vol}(K)$.
2. The value of f on any point $y = (1-\delta)x^* + \delta x \in K^\delta$ is

$$\begin{aligned} f(y) = f((1-\delta)x^* + \delta x) &\leq (1-\delta) f(x^*) + \delta f(x) \leq (1-\delta) f(x^*) + \delta B \\ &\leq f(x^*) + \delta(B - f(x^*)) \leq f(x^*) + 2\delta B. \end{aligned}$$

Using Grünbaum's lemma, the volume falls by a constant factor in each iteration, so $\text{vol}(K_t) \leq \text{vol}(K) \cdot (1 - \frac{1}{e})^t$. If we define $\delta := 2(1 - 1/e)^{T/n}$, then after T steps the volume of K_T is smaller than that of K^δ, so some point of K^δ must have been cut off.

Consider such a step t such that $K^\delta \subseteq K_t$ but $K^\delta \not\subseteq K_{t+1}$. Let $y \in K^\delta \cap (K_t \setminus K_{t+1})$ be a point that is "cut off". By convexity we have

$$f(y) \geq f(c_t) + \langle \nabla f(c_t), y - c_t \rangle;$$

moreover, $\langle \nabla f(c_t), y - c_t \rangle > 0$ since the cut-off point $y \in K_t \setminus K_{t+1}$. Hence the corresponding centroid has value$f(c_t) < f(y) \leq f(x^*) + 2\delta B$. Since $\widehat{x}$ is the centroid with the smallest function value, we get

$$f(\widehat{x}) - f(x^*) \leq 2B \cdot 2\big(1 - 1/e\big)^{T/n}\big) \leq 4B\exp(-T/3n).$$

The second claim follows by substituting $T \geq 3n \ln \frac{4B}{\varepsilon}$ into the first claim, and simplifying. □

18.1.3 *Some Comments on the Runtime*

The number of iterations T to get an error of ε depends on $\log(1/\varepsilon)$; compare this linear convergence to gradient descent requiring $O(1/\varepsilon^2)$ steps. One downside with this approach is that the number of iterations explicitly depends on the number of dimensions n, whereas gradient descent depends on other factors (a bound on the gradient, and the diameter of the polytope), but not explicitly on the dimension.

However, the all-important question is: *how do we compute the centroid*? This is a difficult problem—it is #P-hard to do exactly, which means it is at least as hard as counting the number of satisfying assignments to a SAT instance. In 2002, Bertsimas and Vempala suggested a way to find approximate centroids by sampling random points from convex bodies (which in turn is done via random walks). More details here.

18.2 *The Ellipsoid Algorithm*

The Ellipsoid algorithm is usually attributed to Naum Shor; the fact that this algorithm gives a polynomial-time algorithm for linear programming was a breakthrough result due to Khachiyan, and was front page news at the time. A great source of information about this algorithm is the Gröschel-Lovasz-Schrijver book. A historical perspective appears in this this survey by Bland, Goldfarb, and Todd.

N. Z. Šor and N. G. Žurbenko (1971)

Khachiyan (1979)

Gröschel, Lovasz, and Schrijver (1988)

Let us mention some theorem statements about the Ellipsoid algorithm that are most useful in designing algorithms. The second-most important theorem is the following. Recall the notion of an extreme point or basic feasible solution (bfs) from §7.1.2. Let $\langle A \rangle, \langle b \rangle, \langle c \rangle$ denote the number of bits required to represent of A, b, c respectively.

Theorem 18.3 (Linear Programming in Polynomial Time)**.** *Given a linear program* $\min\{c^\intercal x \mid Ax \geq b\}$*, the Ellipsoid algorithm produces an optimal vertex solution for the LP, in time polynomial in* $\langle A \rangle$, $\langle b \rangle$, $\langle c \rangle$.

One may ask: does the runtime depend on the bit-complexity of the input because doing basic arithmetic on these numbers may require large amounts of time. Unfortunately, that is not the case. Even if we count the number of arithmetic operations we need to perform, the Ellipsoid algorithm performs $\text{poly}(\langle A\rangle + \langle b\rangle + \langle c\rangle)$ operations. A stronger guarantee would have been for the number of arithmetic operations to be $\text{poly}(m, n)$, where the matrix $A \in \mathbb{Q}^{m\times n}$: such an algorithm would be called a ***strongly polynomial-time*** algorithm. Obtaining such an algorithm remains a major open question.

18.2.1 Separation Implies Optimization

In order to talk about the Ellipsoid algorithm, as well as to state the next (and most important) theorem about Ellipsoid, we need a definition.

Definition 18.4 (Strong Separation Oracle)**.** For a convex set $K \subseteq \mathbb{R}^n$, a *strong separation oracle* for K is an algorithm that takes a point $z \in \mathbb{R}^n$ and correctly outputs one of:

(i) `Yes` (i.e., $z \in K$), or

(ii) `No` (i.e., $z \notin K$), as well as a *separating hyperplane* given by $a \in \mathbb{R}^n, b \in \mathbb{R}$ such that $K \subseteq \{x \mid \langle a, x\rangle \leq b\}$ but $\langle a, x\rangle > b$.

The example on the right shows a separating hyperplane.

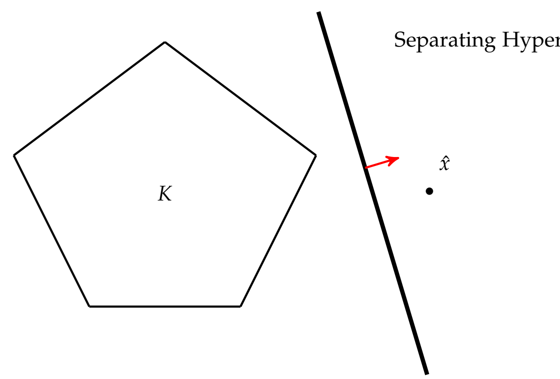

Figure 18.2: Example of separating hyperplanes

Theorem 18.5 (Separation implies Optimization)**.** *Given an LP*

$$\min\{c^\intercal x \mid x \in K\}$$

for a polytope $K = \{x \mid Ax \geq b\} \subseteq \mathbb{R}^n$*, and given access to a strong separation oracle for K, the Ellipsoid algorithm produces a vertex solution for the LP in time* $\text{poly}(n, \max_i\langle a_i\rangle, \max_i\langle b_i\rangle, \langle c\rangle)$.

There is no dependence on the number of constraints in the LP; we can get a basic solution to any finite LP as long as each constraint has a reasonable bit complexity, and we can define a separation oracle for the polytope. This is often summarized by saying: *"separation implies optimization"*. Let us give two examples of exponential-sized LPs, for which we can give a separation oracles, and hence optimize over them. Maybe already discussed earlier?

18.3 Ellipsoid for LP Feasibility

Instead of solving a linear program, suppose we are given a description of some polytope K, and want to either find some point $x \in K$, or to report that K is the empty set. This *feasibility* problem is no harder than optimization over the polytope; in fact, the GLS book shows that feasibility is not much easier than optimization: under certain conditions, the two problems are essentially equivalent to each other. (Very loosely, we can do binary search on the objective function.)

Given this, let's discuss solving a feasibility problem; this will allow us to illustrate some of the major ideas of the Ellipsoid algorithm. Given some description of a polytope K, and two scalars $R, r > 0$, suppose we are guaranteed that

(a) $K \subseteq \text{Ball}(0, R)$, and

(b) either $K = \emptyset$, or else some ball $\text{Ball}(c, r) \subseteq K$ for some $c \in \mathbb{R}^n$.

The feasibility problem is to figure out which of the two cases in condition (b) holds; moreover, if $K \neq \emptyset$ then we also need to find a point $x \in K$. We assume that K is given by a strong separation oracle:

Theorem 18.6 (Idealized Feasibility using Ellipsoid)**.** *Given K, r, R as above (and a strong separation oracle for K), the feasibility problem can be solved using $O(n \log(R/r))$ oracle calls.*

For this result, and the rest of the chapter: let us assume that we can perform ***exact arithmetic on real numbers***. This assumption is with considerable loss in generality, since the algorithm takes square-roots when computing the new ellipsoid. If were to round numbers when doing this, that could create all sorts of numerical problems, and a large part of the complication in the actual algorithms comes from these numerical issues.

Proof. The basic structure is simple, and reminiscent of the Centroid algorithm. At each iteration t, we have a current ellipsoid $\mathcal{E}_t$ guaranteed to contain the set K (assuming we have not found a point $x \in K$ yet). The initial ellipsoid is $\mathcal{E}_0 = \text{Ball}(0, R)$, so condition (a) guarantees that $K \subseteq \mathcal{E}_0$.

In iteration t, we ask the separation oracle whether the center c_t of ellipsoid $\mathcal{E}_t$ belongs to K? If the oracle answers `Yes`, we are done. Else the oracle returns a separating hyperplane $\langle a, x \rangle = b$, such that $\langle a, c_t \rangle > b$ but $K \subseteq H_t := \{x : \langle a, x \rangle \leq b\}$. Consequently, $K \subseteq \mathcal{E}_t \cap H_t$. Moreover, the half-space H_t does not contain the center c_t of the ellipsoid, so $\mathcal{E}_t \cap H_t$ is less than half the entire ellipsoid. The crucial idea is to find another (small-volume) ellipsoid $\mathcal{E}_{t+1}$ containing this piece $\mathcal{E}_t \cap H_t$ (and hence also K). This allows us to continue.

To show progress, we need that the volume $\text{vol}(\mathcal{E}_{t+1})$ is considerably smaller than $\text{vol}(\mathcal{E}_t)$. In §18.5 we show that the ellipsoid $\mathcal{E}_{t+1} \supseteq \mathcal{E}_t \cap H_t$ has volume

$$\frac{\text{vol}(\mathcal{E}_{t+1})}{\text{vol}(\mathcal{E}_t)} \leq e^{-\frac{1}{2(n+1)}}.$$

Therefore, after $2(n+1)$ iterations, the ratio of the volumes falls by at least a factor of $\frac{1}{e}$. Now we are done, because our assumptions say that

This volume reduction is much weaker by a factor of n compared to that of the Centroid algorithm, so it is often worth considering if applications of the Ellipsoid algorithm can be replaced by the Centroid algorithm.

$$\text{vol}(K) \leq \text{vol}(\text{Ball}(0, R)),$$

and that

$$K \neq \emptyset \iff \text{vol}(K) \geq \text{vol}(\text{Ball}(0, r)).$$

Hence, if after $2(n+1)\ln(R/r)$ steps, none of the ellipsoid centers have been inside K, we know that K must be empty. ∎

18.3.1 Finding Vertex Solutions for LPs

There are several issues that we need to handle when solving LPs using this approach. For instance, the polytope may not be full-dimensional, and hence we do not have any non-trivial ball within K. Our separation oracles may only be approximate. Moreover, all the numerical calculations may only be approximate.

Even after we take care of these issues, we are working over the rationals so binary search-type techniques may not be able to get us to a vertex solution. So finally, when we have a solution x_t that is "close enough" to x^*, we need to "round" it and get a vertex solution. In a single dimension we can do the following (and this idea already appeared in a homework problem): we know that the optimal solution x^* is a rational whose denominator (when written in reduced terms) uses at most some b bits. So we find a solution within distance to x^* is smaller than some δ. Moreover δ is chosen to be small enough such that there is a unique rational with denominator smaller than 2^b in the δ-ball around x_t. This rational can only be x^*, so we can "round" x_t to it.

Consider the case where we perform binary-search over the interval $[0, 1]$ and want to find the point $1/3$: no number of steps will get us exactly to the answer.

In higher dimensions, the analog of this is a technique (due to Lovász) called *simultaneous Diophantine equations*. Details here.

18.4 Ellipsoid for Convex Optimization

Now we want to solve $\min\{f(x) \mid x \in K\}$. Again, assume that K is given by a strong separation oracle, and we have numbers R, r such that $K \subseteq \text{Ball}(0, R)$, and K is either empty or contains a ball of radius r. The general structure is a one familiar by now, and combines ideas from both the previous sections.

1. Let the starting point $x_1 \leftarrow 0$, the starting ellipsoid be $\mathcal{E}_1 \leftarrow \text{Ball}(0, R)$, and the starting convex set $K_1 \leftarrow K$.

2. At time t, ask the separation oracle: "Is the center c_t of ellipsoid $\mathcal{E}_t$ in the convex body K_t?"

   Yes: Define half-space $H_t := \{x \mid \langle \nabla f(c_t), x - c_t \rangle \leq 0\}$. Observe that $K_t \cap H_t$ contains all points in K_t with value at most $f(c_t)$.

   No: In this case the separation oracle also gives us a separating hyperplane. This defines a half-space H_t such that $c_t \notin H_t$, but $K_t \subseteq H_t$.

   In both cases, set $K_{t+1} \leftarrow K_t \cap H_t$, and $\mathcal{E}_{t+1}$ to an ellipsoid containing $\mathcal{E}_t \cap H_t$. Since we knew that $K_t \subseteq \mathcal{E}_t$, we maintain that $K_{t+1} \subseteq \mathcal{E}_{t+1}$.

3. Finally, after $T = 2n(n+1)\ln(R/r)$ rounds either we have not seen any point in K—in which case we say "K is empty"—or else we output
$$\widehat{x} \leftarrow \arg\min\{f(c_t) \mid c_t \in K_t, t \in 1 \dots T\}.$$

One subtle issue: we make queries to a separation oracle for K_t, but we are promised only a separation oracle for $K_1 = K$. However, we can build separation oracles for H_t inductively: indeed, given strong separation oracle for K_{t-1}, we build one for $K_t = K_{t-1} \cap H_{t-1}$ as follows:

> Given $z \in \mathbb{R}^n$, query the oracle for K_{t-1} at z. If $z \notin K_{t-1}$, the separating hyperplane for K_{t-1} also works for K_t. Else, if $z \in K_{t-1}$, check if $z \in H_{t-1}$. If so, $z \in K_t = K_{t-1} \cap H_{t-1}$. Otherwise, the defining hyperplane for halfspace H_{t-1} is a separating hyperplane between z and K_t.

Now adapting the analysis from the previous sections gives us the following result (assuming exact arithmetic again):

Theorem 18.7 (Idealized Convex Minimization using Ellipsoid)**.** *Given K, r, R as above (and a strong separation oracle K), and a function $f : K \to [-B, B]$, the Ellipsoid algorithm run for T steps either correctly reports that $K = \emptyset$, or else produces a point $\widehat{x}$ such that*
$$f(\widehat{x}) - f(x^*) \leq \frac{2BR}{r} \exp\left\{-\frac{T}{2n(n+1)}\right\}.$$

Note the similarity to Theorem 18.2, as well as the differences: the exponential term is slower by a factor of $2(n+1)$. This is because the volume of the successive ellipsoids shrinks much slower than in Grünbaum's lemma. Also, we lose a factor of R/r because K is potentially smaller than the starting body by precisely this factor. (Again, this presentation ignores precision issues, and assumes we can do exact real arithmetic.)

18.5 Getting the New Ellipsoid

This brings us to the final missing piece: given a current ellipsoid $\mathcal{E}$ and a half-space H that does not contain its center, we want an ellipsoid $\mathcal{E}'$ that contains $\mathcal{E} \cap H$, and as small as possible. To start off, let us recall some basic facts about ellipsoids. The simplest ellipses in $\mathbb{R}^2$ are axis aligned, say with principal semi-axes having length a and b, and written as:
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} \leq 1.$$
Or in matrix notation we could also say
$$\begin{bmatrix} x \\ y \end{bmatrix}^\intercal \begin{bmatrix} 1/a^2 & 0 \\ 0 & 1/b^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \leq 1$$

More generally, any ellipsoid $\mathcal{E}$ is perhaps best thought of as a invertible linear transformation L applied to the unit ball $B(0,1)$, and then it being shifted to the correct center c. The linear transformation yields:

$$\begin{aligned} L(\text{Ball}(0,1)) &= \{Lx : x^\intercal x \leq 1\} \\ &= \{y : (L^{-1}y)^\intercal (L^{-1}y) \leq 1\} \\ &= \{y : y^\intercal (LL^\intercal)^{-1} y \leq 1\} \\ &= \{y : y^\intercal Q^{-1} y \leq 1\}, \end{aligned}$$

where $Q^{-1} := LL^\intercal$ is a positive semidefinite matrix. For an ellipsoid centered at c we simply write

$$\{y + 1 : y^\intercal Q^{-1} y \leq 1\} = \{y : (y-c)^\intercal Q^{-1} (y-c) \leq 1\}.$$

It is helpful to note that for any ball A,

$$\text{vol}(L(A)) = \text{vol}(A) \cdot |\det(L)| = \text{vol}(A)\ \sqrt{\det(Q)}$$

In the above problems, we are given an ellipsoid $\mathcal{E}_t$ and a half-space H_t that does not contain the center of $\mathcal{E}_t$. We want to find a matrix Q_{t+1} and a center c_{t+1} such that the resulting ellipsoid $\mathcal{E}_{t+1}$ contains $\mathcal{E}_t \cap H_t$, and satisfies

$$\frac{\text{vol}(\mathcal{E}_{t+1})}{\text{vol}(\mathcal{E}_t)} \leq e^{-1/2(n+1)}.$$

Given the above discussion, it suffices to do this when $\mathcal{E}_t$ is a unit ball: indeed, when $\mathcal{E}_t$ is a general ellipsoid, we apply the inverse linear transformation to convert it to a ball, find the smaller ellipsoid for it, and then apply the transformation to get the final smaller ellipsoid. (The volume changes due to the two transformations cancel each other out.)

We give the construction for the unit ball below, but first let us record the claim for general ellipsoids:

Theorem 18.8. *Given an ellipsoid $\mathcal{E}_t$ given by (c_t, Q_t) and a separating hyperplane $a_t^\intercal (x - c_t) \leq 0$ through its center, the new ellipsoid $\mathcal{E}_{t+1}$ with center c_{t+1} and psd matrix Q_{t+1}) is found by taking*

$$c_{t+1} := c_t - \frac{1}{n+1} h$$

and

$$Q_{t+1} = \frac{n^2}{n^2-1} \left(Q_k - \frac{2}{n+1} h h^\intercal \right)$$

where $h = \sqrt{a_t^\intercal Q_t a_t}$.

Note that the construction requires us to take square-roots: this may result in irrational numbers which we then have to either truncate, or represent implicitly. In either case, we face numerical issues; ensuring that these issues are not real problems lies at the heart of the formal analysis. We refer to the GLS book, or other textbooks for details and references.

18.5.1 Halving a Ball

Before we end, we show that the problem of finding a smaller ellipsoid that contains half a ball is, in fact, completely straight-forward. By rotational symmetry, we might as well find a small ellipsoid that contains

$$K = \text{Ball}(0,1) \cap \{x \mid x_1 \geq 0\}.$$

By symmetry, it makes sense that the center of this new ellipsoid $\mathcal{E}$ should be of the form

$$c = (c_1, 0, \ldots, 0).$$

Again by symmetry, the ellipsoid can be axis-aligned, with semi-axes of length a along e_1, and $b > a$ along all the other coordinate axes. Moreover, for $\mathcal{E}$ to contain the unit ball, it should contain the points $(1,0)$ and $(0,1)$, say. So

$$\frac{(1-c_1)^2}{a^2} \leq 1 \quad \text{and} \quad \frac{c_1^2}{a^2} + \frac{1}{b^2} \leq 1.$$

Suppose these two inequalities are tight, then we get

$$a = 1 - c_1, \qquad b = \sqrt{\frac{(1-c_1)^2}{(1-c_1)^2 - c_1^2}} = \sqrt{\frac{(1-c_1)^2}{(1-2c_1}},$$

and moreover the ratio of volume of the ellipsoid to that of the ball is

$$ab^{n-1} = (1-c_1) \cdot \left(\frac{(1-c_1)^2}{1-2c_1}\right)^{(n-1)/2}.$$

This is minimized by setting $c_1 = \frac{1}{n+1}$ gives us fill in details

$$\frac{\text{vol}(\mathcal{E})}{\text{vol}(\text{Ball}(0,1))} = \cdots \leq e^{-\frac{1}{2(n+1)}}.$$

For a more detailed description and proof of this process, see these notes from our LP/SDP course for details.

In fact, we can view the question of finding the minimum-volume ellipsoid that contains the half-ball K: this is a convex program, and looking at the optimality conditions for this gives us the same construction above (without having to make the assumptions of symmetry).

18.6 Algorithms for Solving LPs

While the Centroid and Ellipsoid algorithms for convex programming are powerful, giving us linear convergence, they are not typically used to solve LPs in practice. There are several other algorithms: let us mention them in passing. Let $K := \{x \mid Ax \geq b\} \subseteq \mathbb{R}^n$, and we want to minimize $\{c^\intercal x \mid x \in K\}$.

Simplex: This is perhaps the first algorithm for solving LPs that most of us see. It was also the first general-purpose linear program solver known, having been developed by George Dantzig in 1947. This is a local-search algorithm: it maintains a vertex of the polyhedron K, and at each step it moves to a neighboring vertex without decreasing the objective function value, until it reaches an optimal vertex. (The convexity of K ensures that such a sequence of steps is possible.) The strategy to choose the next vertex is called the ***pivot rule***. Unfortunately, for most known pivot rules, there are examples on which the following the pivot rule takes exponential (or at least a super-polynomial) number of steps. Despite that, it is often used in practice: e.g., the Excel software contains an implementation of simplex.

Interior Point: A very different approach to get algorithms for LPs is via interior-point algorithms: these happen to be good both in theory and in practice. The first polynomial-time interior-point algorithm was proposed by Karmarkar in 1984. We discuss this in the next chapter.

Geometric Algorithms for LPs: These approaches are geared towards solving LPs fast when the number of dimensions n is small. If m is the number of constraints, these algorithms often allow a poor runtime in n, at the expense of getting a good dependence on m. As an example, a randomized algorithm of Raimund Seidel's has a runtime of $O(m \cdot n!) = O(m \cdot n^{n/2})$; a different algorithm of Ken Clarkson (based on the multiplicative weights approach!) has a runtime of $O(n^2 m) + n^{O(n)} O(\log m)^{O(\log n)}$. One of the fastest such algorithm is by Jiri Matoušek, Micha Sharir, and Emo Welzl, and has a runtime of

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

For details and references, see this survey by Martin Dyer, Nimrod Megiddo, and Emo Welzl.

Naturally, there are other approaches to solve linear programs as well: write more here.