The Ellipsoid Algorithm

John E. Mitchell

1 Introduction

The simplex algorithm requires exponentially many iterations in the worst case for known pivot rules. In contrast, the ellipsoid algorithm requires a polynomial number of iterations. The ellipsoid algorithm was developed in the 1970s in the Soviet Union, building on earlier Soviet work on nonlinear programming.

The algorithm is a high-dimensional version of binary search. It can be stated as an algorithm to determine whether a feasible solution exists to a system of inequalities:

Does there exist \( x \in \mathbb{R}^n \) satisfying \( a_i^T x \leq c_i \) for \( i = 1, \ldots, m \)?

We let

\[
P := \{ x \in \mathbb{R}^n : a_i^T x \leq c_i \text{ for } i = 1, \ldots, m \}.
\]

2 Assumptions

We need to make two assumptions:

\textit{Assumption 1}: If \( P \) is nonempty then it includes points contained in a ball of radius \( R \) centered at the origin.

Assumption 1 can be justified through the use of \textit{Cramer’s Rule}, which lets us bound the norm of any extreme point of \( P \).

\textit{Assumption 2}: If \( P \) is nonempty then it contains a ball of radius \( r \).

Assumption 2 is like a nondegeneracy assumption. Cramer’s Rule also lets us place a lower bound on the distance between two extreme points. Technically, we can always make Assumption 2, because the size of the numbers in the parameters can be used to define another parameter \( \delta \) and then each \( c_i \) can be increased by \( \delta \) in such a way that

- the new system has a solution if and only if the old system has a solution, and
- the new system contains a ball of radius \( r \) if it is nonempty.

3 Each Iteration

At each iteration we have an ellipsoid which contains (part of) \( P \), if \( P \) is nonempty. We check the center \( \bar{x} \) of the ellipsoid: does it satisfy

\[a_i^T \bar{x} \leq c_i \text{ for } i = 1, \ldots, m?\]
If YES: we are done.
Otherwise, we have a violated constraint \( a_i^T x \leq c_j \). We define a new ellipsoid that contains the intersection of the old ellipsoid with the halfspace \( \{ x \in \mathbb{R}^n : a_i^T x \leq a_i^T \bar{x} \} \).

If \( n = 1 \) then this would divide our search space in half. In \( \mathbb{R}^n \), we don’t get such a good improvement. Nonetheless, the size of the ellipsoid shrinks enough that we get convergence in a number of iterations polynomial in \( n \).

4 Complexity analysis

Initial ellipsoid: Ball of radius \( R \) centered at the origin.

**How many “Noes” are enough?** If the ellipsoid is too small to contain a ball of radius \( r \) then \( P = \emptyset \). The algorithm updates from an ellipsoid \( E \) to an ellipsoid \( E' \). It can be shown that the volumes of these ellipsoids are related as:

\[
\text{Vol}(E') \leq e^{-1/(2(n+1))}\text{Vol}(E).
\]

How many “Noes” to go from a ball of radius \( R \) to a ball of radius \( r \)? Let \( v \) and \( V \) denote the volumes of balls of radius \( r \) and \( R \), respectively. Let \( E^0 \) be the initial ellipsoid, let \( E^k \) be the ellipsoid after \( k \) iterations. We want the smallest value of \( k \) so that

\[
\text{Vol}(E^k) \leq v.
\]

We also have

\[
\text{Vol}(E^k) \leq \left[ e^{-1/(2(n+1))} \right]^k V = e^{-k/(2(n+1))} V.
\]

Thus it suffices to choose \( k \) sufficiently large that

\[
e^{-k/(2(n+1))} V \leq v.
\]
Equivalently, after dividing by \( V \) and taking logs of both sides, we require
\[-k/(2(n+1)) \leq \ln \left( \frac{v}{V} \right),\]
which can be restated as requiring
\[k \geq 2(n+1) \ln \left( \frac{V}{v} \right).\]
The ratio \( V/v \) can be chosen so that \( \ln(V/v) \) is \( O(n^3) \), so we have an algorithm that requires \( O(n^4) \) iterations.

**Work per iteration:** In exact arithmetic, would need to work with *square roots*. Can show that it is OK to approximate the square roots and still have the volume of the ellipsoid decrease at almost the same rate.

**Finding optimal solutions to linear programs:** If feasible, cut on the objective function. So restrict attention to the halfspace of points at least as good as the current center.

## 5 Separation and optimization

**Definition 1.** Given a convex set \( C \subseteq \mathbb{R}^n \) and a point \( \bar{x} \in \mathbb{R}^n \), the separation problem is to determine whether \( \bar{x} \in C \). If the answer is NO, a separating hyperplane \( a^T x = c \) is determined, where \( a \in \mathbb{R}^n \), \( c \in \mathbb{R} \), \( a^T x \leq c \) for all \( x \in C \), and \( a^T \bar{x} > c \).

**Theorem 1.** Assume the convex set \( C \) contains a ball of radius \( r \). If the separation problem for \( C \) can be solved in polynomial time then the feasibility problem for \( C \) can also be solved in polynomial time.

**Proof.** We use the ellipsoid algorithm. At each iteration, we test whether the center \( x^k \) of the current ellipsoid is in \( C \). If it is not then the separation problem solution returns a valid constraint for \( C \) which we can use to update the ellipsoid.

The overall complexity is the product of two polynomial functions, which is polynomial.

Consider the optimization problem
\[
\min_{x \in \mathbb{R}^n} \quad c^T x \\
\text{subject to} \quad x \in C \subseteq \mathbb{R}^n \quad (CP)
\]
where \( C \) is a convex set and \( c \in \mathbb{R}^n \).

**Theorem 2.** The problem \((CP)\) can be solved in polynomial time if and only if the separation problem for \( C \) can be solved in polynomial time.

This result follows from using the ellipsoid algorithm. We can optimize over \( C \) using the ellipsoid algorithm, generating separating hyperplanes either:
- by solving the separation problem to separate the current center of the ellipsoid from \( C \), or
- by cutting on the objective function if \( x \in C \).
6 Relevance to integer optimization

Consider the integer optimization problem

$$\min_x c^T x \quad \text{subject to} \quad x \in S \subseteq \mathbb{R}^n$$

(IP)

for some subset $S$, and where $Z$ denotes the set of integers. Let $C$ denote the convex hull of the set of integer points in $S$, so

$$C = \text{conv}(S \cap \mathbb{Z}^n)$$

(The convex hull of a set $T$ is the smallest convex set containing $T$. A formal definition will be given next week.) All the extreme points of $C$ are feasible integer points, so we could in principle solve (IP) by solving the linear program

$$\min_x c^T x \quad \text{subject to} \quad x \in C \subseteq \mathbb{R}^n$$

(LP)

The difficulty with this approach is that it is very hard to get an explicit formulation for $C$. If we can solve the separation problem for $C$ in polynomial time then we can solve (LP) in polynomial time, which means we can solve (IP) in polynomial time.

Thus, for $\mathcal{NP}$-complete problems, we cannot expect to solve the separation problem in polynomial time.

Conversely, if we can solve the optimization problem in polynomial time then we can solve the separation problem in polynomial time. For example, we can solve the problem of finding a minimum weight perfect matching on a graph $G = (V, E)$ in polynomial time, and indeed the separation problem of determining whether a point $x$ is in the convex hull of the set of perfect matchings can be solved in $O(|V|^4)$ time.

7 $\mathcal{NP}$-Hard problems

A problem is called $\mathcal{NP}$-hard if it at least as hard as an $\mathcal{NP}$-complete problem, in the sense that the $\mathcal{NP}$-complete problem can be polynomially reduced to the $\mathcal{NP}$-hard problem. An $\mathcal{NP}$-hard problem need not be a feasibility problem. In particular, optimization versions of $\mathcal{NP}$-complete integer programming feasibility problems are $\mathcal{NP}$-hard.