The Ellipsoid Algorithm for Linear Programming

Lecturer: Sanjeev Arora, COS 521, Fall 2005
Princeton University
Scribe Notes: Siddhartha Brahma

The Ellipsoid algorithm for linear programming is a specific application of the ellipsoid method developed by Soviet mathematicians Shor(1970), Yudin and Nemirovskii(1975). Khachiyan(1979) applied the ellipsoid method to derive the first polynomial time algorithm for linear programming. Although the algorithm is theoretically better than the Simplex algorithm, which has an exponential running time in the worst case, it is very slow practically and not competitive with Simplex. Nevertheless, it is a very important theoretical tool for developing polynomial time algorithms for a large class of convex optimization problems, which are much more general than linear programming.

We will start of with a few definitions and then consider the actual algorithm. We will consider general linear programs of the following form defined on vectors in $\mathbf{R}^n$.

$$\begin{aligned} \text{maximize } c^T x \\ Ax \leq b \\ x \geq 0 \end{aligned}$$

where A is a $m \times n$ real constraint matrix and $x, c \in \mathbf{R}^n$.

Definition 1. A **Hyperplane** is defined to be the set of points satisfying the linear equation $ax = b$, where $a, x, b \in \mathbf{R}^n$.

Definition 2. A **Convex Set** $K \subseteq \mathbf{R}^n$ is a set of points such that $\forall x, y \in K$, $\lambda x + (1-\lambda)y \in K$, where $\lambda \in [0, 1]$. A **Convex Body** is a closed and bounded convex set.

A few examples of convex sets and bodies are as follows:
1. Hypercube length l is the set of all x such that $0 \leq x_i \leq l, 1 \leq i \leq n$.
2. Ball of radius r around the origin is the set of all x such that $\displaystyle\sum_{i=1}^{n} x_i^2 \leq r^2$.
3. An axis aligned ellipsoid is the set of all x such that $\displaystyle\sum_{i=1}^{n} x_i^2/a_i^2 \leq 1$ where a_i's are nonzero reals.
4. A general ellipsoid in $\mathbf{R}^n$ can be represented by

$$(x-a)^T B (x-a) \leq 1$$

, where B is a positive semidefinite matrix. (Being positive semidefinite means B can be written as $B = AA^T$ for some $n \times n$ real matrix A. This is equivalent to saying $B = Q^{-1}DQ$, where Q is a unitary and D is a diagonal matrix with all positive entries.)
5. The whole space $\mathbf{R}^n$ is trivially an infinite convex set.

Evidently each of the constraints in a linear program defines half-spaces (which are parts of $\mathbf{R}^n$ bounded on one side by hyperplanes) and the solution lies in the intersection of these half-spaces. It can be shown easily that the intersection of a finite number of half-spaces is a convex set, which may or may not be bounded. We have the following fact about convex sets which is intuitively clear and not hard to prove.

Observation 3. *If $K \subseteq \mathbf{R}^n$ is a convex set and $p \in \mathbf{R}^n$ is a point, then one of the following holds*
(i) $p \in K$
(ii) there is a hyperplane that separates p from K.

This prompts the following definition of a polynomial time Separating Oracle.

Definition 4. A polynomial time **Separating Oracle** for a convex set K is a procedure which given x, either tells that $x \in K$ or returns a hyperplane separating x from K. The procedure should run in polynomial time. Note that from the previous observation, such a plane is guaranteed to exist.

As defined above, the linear programming problem in $\mathbf{R}^n$ with m constraints has inputs given by A, b and an objective vector c. To solve the linear program we need to at least read the whole input. Therefore, if the number of bits to represent the input is L, a polynomial time solution to the problem would mean a running time of $poly(n, m, L)$.

The problem of optimizing an objective function can be reduced to a series of feasibility problems as follows. We start of with an estimate of the maximum value, say c_0 and check for the feasibility of the following system

$$\begin{aligned} c^T x &\geq c_0 \\ Ax &\leq b \\ x &\geq 0 \end{aligned}$$

If the system is infeasible, we know that the optimum is lesser than c_0. We may now decrease c_0, say by a factor of 2 and check for feasibility again. If this is true, we know that the optimum lies in $[c_0/2, c_0)$. This is essentially a binary search to find the optimum with higher accuracies. We get the optimum in a number of steps polynomial in the input size, each step being a call to a feasibility checking algorithm.

Now we can concentrate on the feasibility problem. The **Ellipsoid Algorithm** solves the feasibility problem in an ingenious way. Let us denote the convex set defined by the feasible solution space by S. Further, we assume that the constraints are non-degenerate, so that S is either empty or has a non-zero volumed denoted by $Vol(S)$. In other words we can find a lower bound V_l on $Vol(S)$. We start off with an ellipsoid of volume V_u guaranteed to bound S if it is finite. If $Vol(S)$ is infinite, we start with a suitable V_u and we will eventually get to a feasible point anyway. In our case, the initial bounding ellipsoid is a sphere in $\mathbf{R}^n$. A single step of the algorithm either finds a point in S, in which case we have proved feasibility, or finds another ellipsoid bounding S that has a volume that is substantially smaller than the volume of the previous ellipsoid. We iterate on this new ellipsoid. In the worst case we need to iterate until the volume of the bounding ellipsoid gets below V_l, in which case we can conclude that the system is infeasible. It turns out that only a polynomial number of iterations are required in the case of linear programming. The algorithm does not require an explicit description of the linear program. All that is required is a polynomial time Separating Oracle, which checks whether a point lies in S or not, and returns a separating hyperplane in the latter case. The following high level pseudocode describes the algorithm.

Input: Bounding ellipsoid E_0 for S, Lower bound V_l on $Vol(S)$.
Output: "yes" if the linear program is feasible, "no" otherwise.
Algorithm:

```
i=0;
while($Vol(E_i) \geq V_l$){
  p = Center of $E_i$;
  (ans,$H$) = SepOracle(p);
  if(ans==yes)
    return "yes";
  else{
    Take the separating hyperplane $H$ and let
    $E_{i+1} =$ minimum volume ellipsoid containing $E_i \bigcap H^+$;
    $i = i + 1$;
  }
}
return "no";
```

The running time of the algorithm depends on the complexity of `SepOracle`, the time required to find E_{i+1} and the ratio V_u/V_l. For linear programming, `SepOracle` runs in $O(mn)$ time as all we need to do is check whether p satisfies all the constraints, and return a violating constraint as H(if it exists). The time needed to find E_{i+1} is also polynomial by the following non-trivial lemma from convex geometry.

i

Lemma 5. *The minimum volume ellipsoid surrounding a half ellipsoid (i.e. $E_i \bigcap H^+$ above) can be calculated in polynomial time and*

$$Vol(E_{i+1}) \leq \left(1 - \frac{1}{2n}\right) Vol(E_i)$$

If the `while` loop in the algorithm runs t times, then by the above lemma

$$\left(1 - \frac{1}{2n}\right)^t \leq \frac{V_u}{V_l} \Rightarrow t = O(n \log(V_u/V_l))$$

It can be shown that for a linear program which requires L bits to represent the input, we can choose $V_l = 2^{-c_1 nL}$ and $V_u = 2^{c_2 nL}$ for some constants c_1, c_2, which implies $t = O(n^2 L)$. Therefore, the above algorithm terminates after $O(n^2 L)$ iterations, each iteration taking polynomial time, which gives us an overall running time of $poly(n, m, L)$. For a detailed proof of the above lemma and other derivations, please refer to Santosh Vempala's notes on the course webpage.

Note that the way the algorithm has been presented, it applies to any problem which has a corresponding `SepOracle` procedure. If that procedure runs in polynomial time and we can find suitable V_l, V_u, we will get a polynomial time algorithm for the corresponding problem. These requirements are satisfied by a large number of convex optimization problems which makes the **Ellipsoid Algorithm** very attractive theoretically.