princeton univ. F'14 cos 521: Advanced Algorithm Design

Lecture 17: Oracles, Ellipsoid method and their uses in convex optimization

Lecturer: *Sanjeev Arora* Scribe:

Oracle: *A person or agency considered to give wise counsel or prophetic predictions or precognition of the future, inspired by the gods.*

Recall that Linear Programming is the following problem:

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

where A is a $m \times n$ real constraint matrix and $x, c \in \mathbf{R}^n$. Recall that if the number of bits to represent the input is L, a polynomial time solution to the problem is allowed to have a running time of $poly(n, m, L)$.

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.

In fact we can use it to solve convex optimization problems that are even too large to write down.

1 Linear programs too big to write down

Often we want to solve linear programs that are too large to even write down in polynomial time.

Example 1 Semidefinite programming (SDP) uses the convex set of PSD matrices in $\Re^n$. This set is defined by the following infinite set of constraints: $a^T X a \geq 0 \quad \forall a \in Re^n$. This is really a linear constraint on the X_{ij}'s:

$$\sum_{ij} X_{ij} a_i a_j \geq 0.$$

Thus this set is defined by *infinitely many* linear constraints.

Example 2 (Held-Karp relaxation for TSP) In the traveling salesman problem (TSP) we are given n points and *distances* d_{ij} between every pair. We have to find a salesman

tour, which is a sequence of hops among the points such that each point is visited exactly once and the total distance covered is minimized.

An *integer programming* formulation of this problem is:

$$\begin{aligned} &\min \textstyle\sum_{ij} d_{ij} X_{ij} \\ &X_{ij} \in \{0,1\} \quad \forall i,j \\ \textstyle\sum_{i\in S, j\in \overline{S}} X_{ij} \geq 2 \qquad &\forall S \subseteq V, \quad S \neq \emptyset, V \qquad \text{(subtour elimination)} \end{aligned}$$

The last constraint is needed because without it the solution could be a disjoint union of subtours, and hence these constraints are called *subtour elimination constraints*. The Held-Karp relaxation relaxes the first constraint to $0 \leq X_{ij} \leq 1$. Now this is a linear program, but it has $2^n + n^2$ constraints! We cannot afford to write them down (for then we might as well use the trivial exponential time algorithm for TSP).

Clearly, we would like to solve such large (or infinite) programs, but we need a different paradigm than the usual one that examines the entire input.

2 A general formulation of convex programming

A convex set $\mathcal{K}$ in $\Re^n$ is a subset such that for every $x, y \in \mathcal{K}$ and $\lambda \in [0,1]$ the point $\lambda x + (1-\lambda)y$ is in $\mathcal{K}$. (In other words, the line joining x, y lies in $\mathcal{K}$.) If it is compact and bounded we call it a *convex body*. It follows that if $\mathcal{K}_1, \mathcal{K}_2$ are both convex bodies then so is $\mathcal{K}_1 \cap \mathcal{K}_2$.

A general formulation of convex programming is

$$\begin{aligned} &\min c^T x \\ &x \in \mathcal{K} \end{aligned}$$

where $\mathcal{K}$ is a convex body.

Example 3 Linear programming is exactly this problem where $\mathcal{K}$ is simply the polytope defined by the constraints.

Example 4 Some lectures ago we were interested in semidefinite programming, where $\mathcal{K}$ = set of PSD matrices. This is convex since if X, Y are psd matrices then so is $(X+Y)/2$. The set of PSD matrices is a convex set but extends to ∞. In the examples last time it was finite since we had a constraint like $X_{ii} = 1$ for all i, which implies that $|X_{ij}| \leq 1$ for all i, j. Usually in most settings of interest we can place some *a priori* upper bound on the desired solution that ensures $\mathcal{K}$ is a finite body.

In fact, since we can use binary search to reduce optimization to decision problem, we can replace the objective by a constraint $c^T x \geq c_0$. Then we are looking for a point in the convex body $\mathcal{K} \cap \{x : c^T x \geq c_0\}$, which is another convex body $\mathcal{K}'$. We conclude that convex programming boils down to finding a *single point* in a convex body (where we may repeat this basic operation multiple times with different convex bodies).

Here are other examples of convex sets and bodies.

1. The whole space $\mathbf{R}^n$ is trivially an infinite convex set.
2. Hypercube length l is the set of all x such that $0 \leq x_i \leq l, 1 \leq i \leq n$.
3. Ball of radius r around the origin is the set of all x such that $\sum_{i=1}^{n} x_i^2 \leq r^2$.

2.1 Presenting a convex body: separation oracles

We need a way to work with a convex body $\mathcal{K}$ without knowing its full description. The simplest way to present a body to the algorithm is via a *membership oracle*: a blackbox program that, given a point x, tells us if $x \in \mathcal{K}$. We will work with a stronger version of the oracle, which relies upon the following fact.

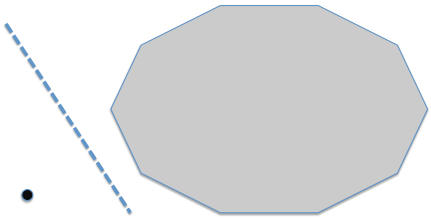

Figure 1: Farkas's Lemma: Between every convex body and a point outside it, there's a hyperplane

Farkas's Lemma: If $\mathcal{K} \subseteq \mathbf{R}^n$ is a convex set and $p \in \mathbf{R}^n$ is a point, then one of the following holds
(i) $p \in \mathcal{K}$
(ii) there is a hyperplane that separates p from $\mathcal{K}$. (Recall that a hyperplane is the set of points satisfying a linear equation of the form $ax = b$ where $a, x, b \in \mathbf{R}^n$.)

This Lemma is intuitively clear but the proof takes a little formal math and is omitted.

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

Definition 1 *A polynomial time* **Separation Oracle** *for a convex set $\mathcal{K}$ is a procedure which given p, either tells that $p \in K$ or returns a hyperplane that separates p and all of K. The procedure runs in polynomial time.*

Example 5 Consider the polytope defined by the Held-Karp relaxation. We are given a candidate solution $P = (P_{ij})$. Suppose $P_{12} = 1.1$. Then it violates the constraint $X_{12} \leq 1$, and thus the hyperplane $X_{12} = 1$ separates the polytope from P.

Thus to check that it lies in the polytope defined by all the constraints, we first check that $\sum_j P_{ij} = 2$ for all i. This can be done in polynomial time. If the equality is violated for any i then that is a separating hyperplane.

If all the other constraints are satisfied, we finally turn to the subtour elimination constraints. We construct the weighted graph on n nodes where the weight of edge $\{i, j\}$ is P_{ij}. We compute the minimum cut in this weighted graph. The subtour elimination

constraints are all satisfied iff the minimum cut $S, \overline{S}$ has capacity ≥ 2. If the mincut $S, \overline{S}$ has capacity less than 2 then the hyperplane

$$\sum_{i \in S, j \in \overline{S}} X_{ij} = 2,$$

has P on the < 2 side and the Held-Karp polytope on the ≥ 2 side.

Thus you can think of a separation oracle as providing a "letter of rejection"to the point outside it explaining why it is not in the body K.

EXAMPLE 6 For the set of PSD matrices, the separation oracle is given a matrix P. It computes eigenvalues and eigenvectors to check if P only has nonnegative eigenvalues. If not, then it takes an eigenvector a corresponding to a negative eigenvalue and returns the hyperplane $\sum_{ij} X_{ij} a_i a_j = 0$. (Note that a_i's are constants here.) Then the PSD matrices are on the ≥ 0 side and P is on the < 0 side.

3 Ellipsoid Method

The Ellipsoid algorithm solves the basic problem of finding a point in a convex body $\mathcal{K}$. The basic idea is *divide and conquer*. At each step the algorithm asks the separation oracle about a particular point p. If p is in $\mathcal{K}$ then the algorithm can declare success. Otherwise the algorithm is able to divide the space into two (using the hyperplane provided by the separation oracle) and recurse on the correct side. (To quote the classic GLS text: *How do you catch a lion in the Sahara? Fence the Sahara down the middle. Gaze on one side and see if you spot the lion on the left. If so, continue on the left side, else continue on the right.*)

The only problem is to make sure that the algorithm makes progress at every step. After all, space is infinite and the body could be anywhere it. Cutting down an infinite set into two still leaves infinite sets. For this we use the notion of the *containing Ellipsoid* of a convex body.

An *axis aligned ellipsoid* is the set of all x such that

$$\sum_{i=1}^{n} \frac{x_i^2}{\lambda_i^2} \leq 1,$$

where λ_i's are nonzero reals. in $3D$ this is an egg-like object where a_1, a_2, a_3 are the radii along the three axes (see Figure 2). A *general ellipsoid* in $\mathbf{R}^n$ can be represented as

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

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 $B = Q^{-1} D Q$, where Q is a unitary and D is a diagonal matrix with all positive entries.)

The convex body $\mathcal{K}$ is presented by a membership oracle, and we are told that the body lies somewhere inside some ellipsoid E_0 whose description is given to us. At the ith iteration

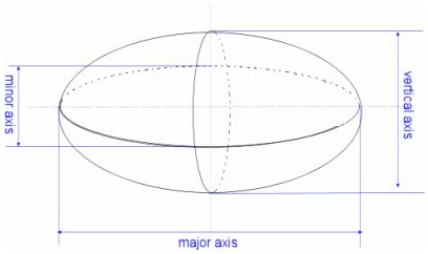

Figure 2: 3D-Ellipsoid and its axes

algorithm maintains the invariant that the body is inside some ellipsoid E_i. The iteration is very simple.

Let p = central point of E_i. Ask the oracle if $p \in \mathcal{K}$. If it says "Yes," declare succes. Else the oracle returns some halfspace $a^T x \geq b$ p that contains $\mathcal{K}$ whereas p lies on the other side. Let E_{i+1} = minimum containing ellipsoid of the convex body $E_i \cap \left\{x : a^T x \geq b\right\}$.

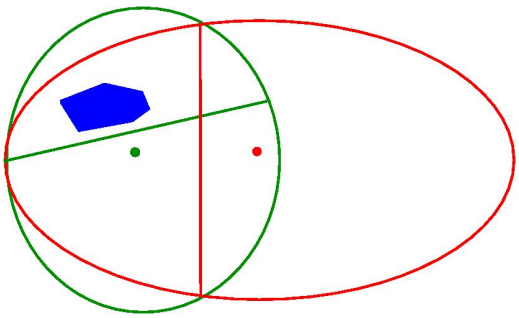

Figure 3: Couple of runs of the Ellipsoid method showing the tiny convex set in blue and the containing ellipsoids. The separating hyperplanes do not pass through the centers of the ellipsoids in this figure.

The running time of each iteration depends on the running time of the separation oracle and the time required to find E_{i+1}. For linear programming, the separation oracle 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 the halfspace (if it exists). The time needed to find E_{i+1} is also polynomial by the following non-trivial lemma from convex geometry.

Lemma 1
The minimum volume ellipsoid surrounding a half ellipsoid (i.e. $E_i \bigcap H^+$ where H^+ is a halfspace as above) can be calculated in polynomial time and

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

Thus after t steps the volume of the enclosing ellipsoid has dropped by $(1 - 1/2n)^t \leq \exp(-t/2n)$.

Technically speaking, there are many fine points one has to address. (i) The Ellipsoid method can never say unequivocally that the convex body was empty; it can only say after T steps that the volume is less than $\exp(-T/2n)$. In many settings we know a priori that the volume of $\mathcal{K}$ if nonempty is at least $\exp(-n^2)$ or some such number, so this is good enough. (ii) The convex body may be low-dimensional. Then its n-dimensional volume is 0 and the containing ellipsoid continues to shrink forever. At some point the algorithm has to take notice of this, and identify the lower dimensional subspace that the convex body lies in, and continue in that subspace.

As for linear programming can be shown that for a linear program which requires L bits to represent the input, it suffices to have volume of $E_0 = 2^{c_2 nL}$ (since the solution can be written in $c_2 nL$ bits, it fits inside an ellipsoid of about this size) and to finish when volume of $E_t = 2^{-c_1 nL}$ for some constants c_1, c_2, which implies $t = O(n^2 L)$. Therefore, the after $O(n^2 L)$ iterations, the containing ellipsoid is so small that the algorithm can easily "round" it to some vertex of the polytope. (This number of iterations can be improved to $O(nL)$ with some work.) Thus the overall running time is $poly(n, m, L)$. For a detailed proof of the above lemma and other derivations, please refer to Santosh Vempala's notes linked from the webpage. The classic [GLS] text is a very readable yet authoritative account of everything related (and there's a lot) to the Ellipsoid method and its variants.

BIBLIOGRAPHY

[GLS] M. Groetschel, L. Lovasz, A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer 1993.