princeton univ. F'17 cos 521: Advanced Algorithm Design

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

Lecturer: *Matt Weinberg* Scribe:*Sanjeev Arora*

**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 (or for that matter, too big to fit into all the hard drives of the world).

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 (cycle), 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:

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

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*.[1] 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).

Example 3 (Dual of Configuration LP) We saw the configuration LP on Homework 2 Extra Credit (it also has other uses, btw). It's dual is the following:

$$\begin{gathered}\min \textstyle\sum_i u_i + \sum_j p_j \\ u_i \geq v_i(S) - \textstyle\sum_{j \in S} p_j \forall S \subseteq 2^{[m]} \\ p_j, u_i \geq 0.\end{gathered}$$

Where there are $n$ $u_i$ variables and $m$ $p_j$ variables, and $2^m$ constraints. So the number of variables is again polynomial, but there are exponentially many constraints.

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

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

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

[1] Exercise: come up with an example showing that the subtour elimination constraints are necessary (i.e. without them, the IP admits non-tour solutions). Another exercise: come up with an example showing that the previous constraints ($\sum_j X_{ij} = 2$) cannot be relaxed to $\sum_j X_{ij} \geq 2$ and folded into the subtour elimination constraints.

EXAMPLE 4 Linear programming is exactly this problem where $\mathcal{K}$ is simply the polytope defined by the constraints.

EXAMPLE 5 In the last lecture 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 $\infty$. 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 testing a convex body for emptiness (i.e., whether it has any point in it).

*Find a point in $\mathcal{K}$ (if such a point exists),*

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

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 and bounding boxes

Since we are talking about solving LPs too large to even write down, 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 black-box 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.

**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 (Proof in Lecture 7 on Strong Duality).

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.*

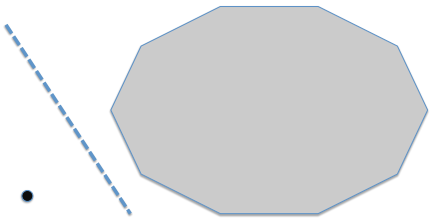

Figure 1: Farkas's Lemma: Between every convex body and a point outside it, there's a hyperplane

EXAMPLE 6 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 $\geq 2$. 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 $\geq 2$ 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 7 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 $\geq 0$ side and $P$ is on the $< 0$ side.

EXAMPLE 8 For the dual of the configuration LP, we can't necessarily find a separating hyperplane in poly-time, depending on how we have access to the valuations $v_i(\cdot)$. If our access allows us to compute $\max_{S \subseteq [m]} v_i(S) - \sum_j p_j$, then we can find whether any of the constraints are violated or not ($u_i$ just needs to be $\geq \max_{S \subseteq [m]} v_i(S) - \sum_j p_j$). This is called a *demand oracle*, and corresponds to thinking of $v_i(\cdot)$ as a valuation function over sets of items, and picking the set that maximizes the buyer's value minus price.

A separation oracle is not sufficient to allow the algorithm to test the body for nonemptiness in finite time. Each time the algorithm questions the oracle about a point $x$, the oracle

could just answer $x \notin \mathcal{K}$, since the convex body could be further from the origin than *all* the (finitely many) points that the algorithm has queried about thus far. After all, space is *infinite*!

Thus the algorithm needs some very rough idea of where $\mathcal{K}$ may lie. It needs $\mathcal{K}$ to lie in some known *bounding box*. The bounding box could be a cube, sphere etc. For example, in the TSP case we see that all $X_{ij}$ lie in $[0, 1]$, which means that the polytope lies in the unit cube.

The Ellipsoid method will use an ellipsoid as a bounding box.

## 3 Ellipsoid Method

The Ellipsoid algorithm solves the basic problem of finding a point (if one exists) 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. Wait for a passerby and ask which side the lion is on. Then continue on that side of the fence. Do this until you've found the lion, or the fenced area is too small to contain a lion in which case you know there was no lion to begin with.*

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. To ensure progress 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 $\lambda_i$'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 $B = Q^{-1}DQ$, 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 $i$th iteration 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 $E_i \cap \left\{x : a^T x \geq b\right\}$.*

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

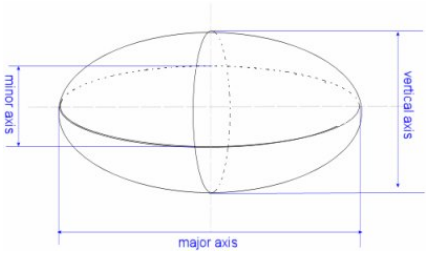


Figure 2: 3D-Ellipsoid and its axes

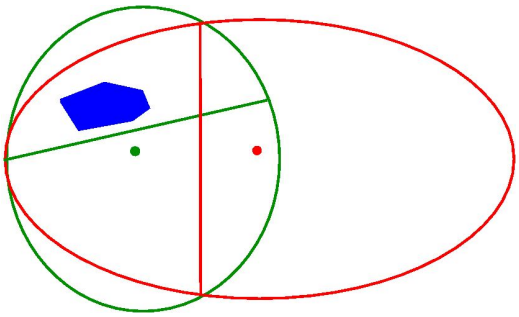

Figure 3: Couple of runs of the Ellipsoid method showing the tiny convex set in blue and the containing ellipsoids. The separating hyperplanes do not pass through the centers of the ellipsoids in this figure.

$O(mn)$ 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.

To sum up, the importance of the Ellipsoid method is that it allows you to see *at a glance* that a convex optimization problem is solvable in polynomial time: (a) Is there a polynomial-time separation oracle? (b) Can we give a rough idea of where the body lies: give a bounding ellipsoid whose volume is only $\exp(poly(n))$ times the volume of the body (assuming the body is nonempty)?

Under these minimal conditions, the problem can be solved in polynomial time!

## 4 Equivalence of Separation and Optimization

We just saw the following: given a separation oracle for a convex region $\mathcal{K}$, we can optimize linear functions over $\mathcal{K}$ (using the Ellipsoid algorithm). It turns out that these two problems are computationally equivalent, due to a famous result of Grotschel, Lovasz, and Schrijver, and independently Karp and Papadimitriou. To be clear:

- **Separate($\mathcal{K}$)**: Given as input a vector $\vec{x}$, output yes if $\vec{x} \in \mathcal{K}$, or a hyperplane separating $\vec{x}$ from $\mathcal{K}$.
- **Optimize($\mathcal{K}$)**: Given as input a vector $\vec{w}$, output $\arg\max_{\vec{x} \in \mathcal{K}} \{\vec{x} \cdot \vec{w}\}$.

The Ellipsoid algorithm shows that if one has a poly-time algorithm for Separate($\mathcal{K}$), then one has a poly-time algorithm for Optimize($\mathcal{K}$) as well. We now show how to take a poly-time algorithm for Optimize and use it to get a poly-time algorithm for Separate.

Think of a separation oracle as finding the "most violated" hyperplane. If no such hyperplane exists, we simply output "yes." It turns out that this problem can be phrased as a linear program:

$$\max \sum_i w_i x_i$$
$$\sum_i w_i y_i \leq 1 \quad \forall y \in \mathcal{K}$$

If the value of this LP is $> 1$, then we have explicitly found a $w$ such that $w \cdot x > w \cdot y$ for all $y \in \mathcal{K}$. Also observe that if $x \notin \mathcal{K}$, there exists a $w$ such that $\sum_i x_i w_i > 1 = \max_{y \in \mathcal{K}} \sum_i w_i y_i$

is feasible for the LP and has value $> 1$ (by the separating hyperplane theorem). So we get that $x \in \mathcal{K}$ if and only if the value is $> 1$, and if the value is $> 1$, $w$ is our separating hyperplane. So if we can solve this LP we have a separation oracle.

Now observe that we can solve this LP as long as we have a separation oracle to determine whether any constraint of the form $\sum_i y_i \leq 1$ is violated. But this is easy! For a given $w$, we just need to find $\max_{y \in \mathcal{K}}\{\sum_i w_i y_i\}$. If this is $> 1$, then we have found a violated constraint. If this is $\leq 1$, then we k now that $\sum_i w_i y_i \leq 1$ for all $y \in \mathcal{K}$.

So to recap: if we can solve this LP, we can get a separation oracle for $\mathcal{K}$. The ellipsoid algorithm let's us solve this LP as long as we can get a separation oracle for the space of $w$ such that $\sum_i y_i w_i \leq 1$ for all $y \in \mathcal{K}$ (also called the *polar* of $\mathcal{K}$). We can get a separation oracle for the polar as long as we can optimize over $\mathcal{K}$. So we can get a separation oracle for $\mathcal{K}$ whenever we can optimize over $\mathcal{K}$.

BIBLIOGRAPHY

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

[GLS 1981 ]M. Groetschel, L. Lovasz, A. Schrijver. *The Ellipsoid Algorithm and its consequences in combinatorial optimization.* Combinatorica, 1981.

[KP 1980 ] R. M. Karp, C. H. Papadimitriou. *On Linear Characterizations of Combinatorial Optimization Problems.* Foundations of Computer Science, 1980.