The Ellipsoid Algorithm

Uri Feige

November 2011

1 The Ellipsoid Algorithm

The Ellipsoid algorithm was developed by (formerly) Soviet mathematicians (Shor (1970), Yudin and Nemirovskii (1975)). Khachian (1979) proved that it provides a polynomial time algorithm for linear programming. The average behavior of the Ellipsoid algorithm is too slow, making it not competitive with the simplex algorithm. However, the theoretical implications of the algorithm are very important, in particular, providing the first proof that linear programming (and a host of other problems) are in P.

The following lecture notes accompanied a course given in 2008. For the 2011 course, this subject was covered in far less details (mostly the geometric view). These notes can serve as supplementary reading for interested students.

1.1 Some preprocessing

Recall that we have seen that if there is a polynomial time algorithm for finding a feasible solution to linear programs, then there is a polynomial time algorithm for finding optimal solutions. Hence we shall only be concerned with finding a feasible solution to an LP.

Recall also that we saw that feasible LPs in standard form always have a basic feasible solution. Moreover, if the input (entries of A, b) consists of integers bounded by p, then every bfs can be expressed as rational numbers whose numerators and denominators are bounded in absolute value by $m!p^m$. (If the inputs are rational numbers, then we can multiply each entry by the least common multiple of all denominators (or just by their product) to get an integer input.)

Now modify the LP by adding some slackness to each inequality. That is, for some small ϵ, change every inequality $\sum a_i x_i \leq b_j$ to $\sum a_i x_i \leq b_j + \epsilon$. Include in this process all nonnegativity constraints, and all equalities (by treating them as a pair of inequalities). It can be showed that when ϵ is small enough (e.g., $\epsilon < (m!p^m)^{-2}$), then the following holds:

1. If the original LP is infeasible, then the new LP is infeasible.
2. If the original LP is feasible, then the feasible region of the new LP contains a full dimensional ball or radius ϵ. (Moreover, it can be shown that given any point at distance at most ϵ from a vertex of the original LP, one can "jump" in polynomial time to the vertex. The method to do this uses *continued fraction* approximation of real numbers by rational numbers with small denominators.)

Scaling the new LP by a factor of $1/\epsilon$ we are left with the following problem. There is an LP defining a polyhedron that is either empty, or contains a full dimensional ball of radius 1. Moreover, there is a bounding ball of radius r (taking $r = m!p^m/\epsilon$ suffices) and centered at the origin that contains this unit ball. We want to design a polynomial time algorithm that on every such input either finds a feasible solution to the LP, or (correctly) reports that the LP is infeasible. By the discussion above, this will give a polynomial time algorithm for linear programming.

Observe that $\log r$ is polynomial in the original parameters of the problem (in n, m and $\log p$). Hence the running time of our algorithm is allowed to be polynomial in $n, m, \log p, \log r$.

1.2 A geometric search problem

In Section 1.1 we reduced the problem of solving linear programs to the following geometric problem. A polyhedron P in R^n is given as a system of linear inequalities. It is known that the intersection of P with a ball $S(0,r)$ contains a a unit ball. Find a point in $P \bigcap S(0,r)$. (A note on notation. Often $S(x,r)$ denotes a sphere of radius r centered at x (a ball without its interior), whereas $B(x,r)$ denotes a ball. However, we reserve the letter B for future use, and hence denote balls by S.)

The following algorithm can be used to solve this problem. Pick a point $y \in S(0,r)$. If $y \in P$ (this can be determined by checking whether y satisfies the system of linear inequalities) then we are done. If not, then some linear inequality is violated. This means that we are given a hyperplane such that P lies entirely within one side of the hyperplane. Equivalently, we are given a half space H that contains P. Moreover, $S(0,r) \bigcap H \bigcap P$ contains a unit ball. Now we can limit the search for a point in P to only $S(0,r) \bigcap H$, which is smaller than $S(0,r)$. Picking a new point y in this smaller region we recurse.

In analyzing the search procedure given above, we assume that we choose y in an oblivious way, without inspecting the system of inequalities. One reason to do so is that in general, it is not known how to make good use of the information provided by the system of inequalities in order to come up with a better choice of y. Another reason is that this oblivious choice makes the eventual algorithm applicable in a wider range of instances, as will be discussed in Section 1.5.

Abstracting the search problem further, we are in the following situation. Unit ball U is hidden within a ball of radius r. We wish to find a point within U. In each step of the algorithm we can choose a point y and get one of the two following answers: either $y \in U$ (and then we are done), or we get an open halfspace H not containing y such that $U \subset H$. We wish to design a strategy for finding a point in U as quickly as possible.

This search problem somewhat resembles binary search. Indeed, we can choose the origin (the center of $S(0,r)$) as our first query point, and then regardless of the halfspace returned, the search space is decreased by a factor of 2. However, starting from the second iteration (at which point the search region is no longer a ball, and hence looses its symmetry), we encounter some difficulties.

1. Is there any point y for which regardless of the halfspace returned, the size of the search space decreases by a factor of 2?

2. If not, then by how much can we decrease the search space by a clever choice of a point y?

3. Is there an efficient algorithm for finding such a point y?

The answer to the first question is unfortunately no. For example, it can be shown that no matter what point we choose in a triangle, there is always an open halfspace not containing this point whose intersection with the triangle contains a 5/9-fraction of the area of the triangle. (This is left as homework. As a hint, see the proof of Proposition 2.)

To the second question we have two answers.

Proposition 1 *For any (measurable) set S in R^d, there is always a point y (not necessarily in S) such that for any halfspace H not containing y, the volume of $H \bigcap S$ is at most $d/(d+1)$ times the volume of S.*

Proof: Recall that in a previous homework we proved that for every collection of halfspaces in R^d, if every $d+1$ of them intersect, then they all intersect at some point. Now consider all halfspaces that contain more than a $d/(d+1)$ fraction of the volume of S. Every $d+1$ of them intersect at some point in S, hence they all intersect. Let y be a point in their intersection. Hence no halfspace not containing y contains more than a $d/(d+1)$ fraction of the volume of S. $\square$

The above proposition is best possible (by considering the $d+1$ vertices of a simplex as the set S). However, in can be strengthened when S is convex.

Proposition 2 *For any convex set S in R^d, there is always a point y (within S) such that for any halfspace H not containing y, the volume of $H \bigcap S$ is at most $1 - 1/e$ times the volume of S.*

Proof: We only sketch the proof. Choose y as the center of gravity of S (which is the same as the average location of a random point in S). Consider an arbitrary hyperplane H through y. Let z be the normal to this hyperplane. Then y is also the center of gravity of the projection of S on z. Using the fact that S is convex, a shifting argument shows that the worst case distribution of weights of the projection on z behaves like z^{d-1}, for z in the range 0 to t. The total volume in this case is $V = \int_0^t z^{d-1}dz = t^d/d$. The center of gravity in this case is $\frac{1}{V}\int_0^t z \cdot z^{d-1}dz = \frac{d}{d+1}t$. The volume of the smaller side of the hyperplane is $V = \int_0^{dt/(d+1)} z^{d-1}dz = (1 - \frac{1}{d+1})^d V > V/e$. $\square$

In our search problem all regions are convex. It follows that by taking y as the center of gravity, in each iteration we cut the search space by a constant factor. In $O(d \log r)$ steps, the volume of the search space becomes smaller than that of a unit ball, implying that by this time we are done.

We now reach the third question mentioned above. It appears that finding the center of gravity of a general convex region is a costly operation. In principle, starting from a point within the convex region, a good enough approximation for the center of gravity can be found by a randomized algorithm. (See for example the STOC2002 paper of Bertsimas and Vempala). However, this was not known at the time that the Ellipsoid algorithm was developed.

The Ellipsoid algorithm uses a less efficient search procedure (in terms of number of iterations), with the advantage that each individual iteration is easier to implement.

1.3 Using ellipsoids

For any set in R^d that is symmetric in the sense that if it contains point x then it also contains the point $-x$, the origin is the center of gravity, and moreover, every hyperplane through the origin divides it into equal size parts. We call a set symmetric if it is symmetric in the above sense when its center of gravity is translated to the origin. The ellipsoid algorithm keeps the search region (for the geometric search problem of Section 1.2) symmetric. After cutting the search region in two, it expands it in a careful way to regain symmetry, while still maintaining some decrease in volume.

Let Q be an n by n nonsingular real matrix and $t \in R^n$. The mapping $T(x) = Qx + t$ is called an *affine transformation.*

A *unit ball* $S(0, 1)$ in R^n is the set $\{x|x^T x \leq 1\}$.

An *ellipsoid* is the image of a unit ball under an affine transformation.

Observe that $y = Qx + t$ implies $x = Q^{-1}(y - t)$. Hence an ellipsoid is

$$T(S(0,1)) = \{y|(Q^{-1}(y-t))^T Q^{-1}(y-t) \leq 1\} = \{y|((y-t)^T B^{-1}(y-t) \leq 1\}$$

where $B = QQ^T$.

The matrix B is *positive definite* meaning that it is real and symmetric, and satisfies the conditions in the following lemma.

Lemma 3 *Given a real symmetric matrix B, the following conditions (defining the matrix to be* positive definite*) are equivalent.*

1. *$x^T Bx > 0$ for all nonzero $x \in R^n$.*
2. *all its eigenvalues are real and positive.*
3. *all upper left submatrices have positive determinants.*
4. *there exists a real matrix Q with linearly independent rows such that $B = QQ^T$.*

Proof: First let us note that the eigenvalues of B are necessarily real. Consider an arbitrary (possibly complex) eigenvector v for B of norm 1, let v^* denote is conjugate transpose, and let λ denote its (supposedly complex) eigenvalue. Let us compute $v^*Bv = v^*\lambda v = \lambda$. Using the symmetry of B we also have $v^*Bv = (B^T v)^* v = (\lambda v)^* v = v^* \lambda^* v = \lambda^*$. Hence λ is real. Likewise, there is no point in taking the corresponding eigenvector v as complex, because the matrix B is real, and then the real and imaginary parts of v never mix. As λ is real, we get that the complex eigenvector is just a combination of a real vector and an imaginary vector, each with the same eigenvalue.

Condition 1 implies condition 2. Otherwise, take x to be an eigenvector with nonpositive eigenvalue.

Condition 2 implies condition 4. Observe that the eigenvectors of B can be taken as orthonormal. For eigenvectors sharing the same eigenvalue, there is always such a choice. (Remark: for nonsymmetric matrices this is not always possible. Consider for example an order 2 matrix whose first row is $(1, 1)$ and second row is $(0, 2)$. Its eigenvectors $(1, 0)$ and $(1, 1)$ are not orthogonal.) For eigenvectors v_i and v_j with different eigenvalues λ_i and λ_j, observe that by symmetry of B we have that $\lambda_j v_i^T v_j = v_i^T B v_j = \lambda_i v_i^T v_j$, implying that

$v_i^T v_j = 0$. One possible choice for Q is to have column i equal to $\sqrt{\lambda_i} v_i$, where v_i is the ith eigenvector of B, and λ_i is its eigenvalue. This can be verified to be correct by comparing the effect of multiplying by the eigenvectors.

Condition 4 implies condition 1. Because $x^T B x = x^T Q Q^T x = (Q^T x)^T (Q^T x)$ which is a nonzero square (because Q has full rank).

We shall not prove here the equivalence of condition 3 to the other conditions, and we do not plan to use condition 3. □

The eigenvectors of B are the principle axes of the ellipsoid, the square roots of the eigenvalues are their lengths, and the square root of the determinant gives the volume (scaled by the volume of the unit ball).

In the ellipsoid algorithm we construct a sequence of ellipsoids $E_k = (B_k, t_k)$. If t_k violates the constraint $a_i^T x \le b_i$ then we take E_{k+1} to be an ellipsoid that contains $\frac{1}{2}E_k = \{y \in E_k : a_i^T y \le a_i^T t_k\}$. There is slackness here as $a_i^T t_k$ is larger than b_i, and the purpose of this slackness is only so as to make the formulas simpler. For E_{k+1} as above there are the following formulas:

$$t_{k+1} = t_k - \frac{1}{n+1} \frac{B_k a_i}{\sqrt{a_i^T B_k a_i}}$$

$$B_{k+1} = \frac{n^2}{n^2-1}(B_k - \frac{2}{n+1} \frac{B_k a_i a_i^T B_k}{a_i^T B_k a_i})$$

It can be shown that $\mathrm{vol}(E_{k+1}) < e^{-1/2(n+1)} \mathrm{vol}(E_k)$. The proof of this last statement, as well as the derivation of the formula above for E_{k+1}, can be based on the following principles. First, everything is proved assuming that E_k is the simplest ellipsoid, namely, the unit ball. Thereafter, the formula for E_{k+1} for general E_k is obtained by using linear transformations. The ratio of volumes remains unchanged by these transformations.

1.4 Numerical precision

In general, one cannot represent the Ellipsoids exactly by rational numbers. The term $\sqrt{a_i^T B_k a_i}$ in the expression for t_{k+1} will in general be irrational. (Do not be confused by the fact that B_k is Positive Definite. It does imply that that $a_i^T B_k a_i$ can be represented as a sum of squares, but not as a single square. Hence its square root might be irrational.) Hence the ellipsoid algorithm cannot be run as described. Instead, to be able to use rational number of bounded numerators, one can take a slightly larger E_{k+1} than the minimum needed, provided it still has (significantly) smaller volume than E_k. This last condition implies that the numerical precision required in order to run the ellipsoid algorithm is pretty high, which contributes to the algorithm not being considered practical.

1.5 Separation oracles

Observe that the number of iterations used by the ellipsoid algorithm depends on the ratio of the bounding ball and the bounded ball. We expressed this ratio as a function of m, the total number of constraints, but in fact it depends not on the total number of constraints, but rather on the number of constraints that are tight in a basic feasible solution. This number never needs to exceed n, the number of variables. Hence beyond some point, adding more constraints to the linear program (even infinitely many constraints, in principle) does not effect the bound on the number of iterations. It may only effect the complexity of implementing a single iteration of the algorithm, namely that of finding a violated constraint. The task of finding a violated constrain given a non-feasible point is often referred to as a *separation oracle* (since the violated constraint is a hyperplane that separates the non-feasible point from the set of feasible solutions).

As a consequence, the ellipsoid algorithm can serve to find optimal or near optimal solutions not just to linear programs, but also to a wider class of optimization problems. We present such as example in the following section.

1.6 Low distortion embeddings

Consider the following problem. The input is an n by n nonnegative symmetric matrix $D = \{d_{ij}\}$, with $d_{ii} = 0$ along the diagonal. One seeks to find a set of n points in R^n whose Euclidean distance matrix is exactly D. Namely, the distance between point i and point j needs to be d_{ij}. This is sometimes referred to as an isometric embedding of the finite metric D in R^n (with ℓ_2 norm). In general, even if the given D is a valid distance matrix (satisfying the triangle inequality), it may not correspond to a set of Euclidean distances. One such example is $d_{12} = d_{13} = d_{14} = 1$ and $d_{23} = d_{34} = d_{24} = 2$.

There is a natural algorithm for checking if the finite metric space embeds in Euclidean space. Place the first point at the origin. Thereafter, place any new point j at an arbitrary point of intersection of the spheres centered at the locations of previous points $i < j$ and of respective radii d_{ij}, if such a point exists. By symmetry arguments, it does not matter which point of intersection is chosen. This process will successfully embed all points if and only if they are embedable in ℓ_2 (assuming computations with arbitrary precision).

We now consider a more general problem in which distances are not given exactly, but instead upper bounds d^+_{ij} and lower bounds d^-_{ij} are given on distances, and the goal is to find a Euclidean embedding satisfying these constraints. This may be the appropriate question when computations are done with finite precision (as exact irrational distances may be too difficult to perform arithmetic on), or when distance measures are given only approximately, or when the finite metric space does not embed in ℓ_2 and one wishes to minimize the distortion, and in a host of other applications.

We now show how questions of the above type can be solved in polynomial time up to a small error term.

For every i, let x_i be vector variable representing the embedding of point i in R^n. Let x_{ij} serve as short hand notation for the inner product $\langle x_i, x_j \rangle$. One can replace the constraints on $|x_i - x_j|$ by equivalent constraints on $(x_i - x_j)^2 = x_{ii} - 2x_{ij} + x_{jj}$. Namely, the constraints are:

$$(d^-_{ij})^2 \leq x_{ii} - 2x_{ij} + x_{jj} \leq (d^+_{ij})^2 \tag{1}$$

Rather than viewing the x_i as variables, view the x_{ij} as variables. Then the constraints (1) are linear in these variables, and hence we see that the set of inner products need to satisfy linear constraints. Finding a set of values x_{ij} satisfying these constraints can in principle be done using linear programming. The problem that remains is that the values

x_{ij} found by the solution of the linear program need not correspond to inner products of vectors. To overcome this issue, we consider the matrix $X = \{x_{ij}\}$. Recall that by item (4) in Lemma 3, if a matrix is positive definite, there are vectors (the rows of Q) such that the entries of the matrix are the inner products of the corresponding vectors. Moreover, these vectors are linearly independent. In our case, we do not need the vectors representing the embeddings of points to be linearly independent, and hence it will suffice to consider positive semidefinite matrices.

Lemma 4 *Given a real symmetric matrix B, the following conditions (defining the matrix to be* positive semidefinite*, psd) are equivalent.*

1. $x^T Bx \geq 0$ *for all* $x \in R^n$.
2. *all its eigenvalues are real and nonnegative.*
3. *there exists a real matrix* Q *such that* $B = QQ^T$.

We omit the proof of Lemma 4 due to its similarity to the proof of Lemma 3.

We now represent the embedding question as a feasibility question for a *positive semidefinite program* (SDP) on $n(n+1)/2$ variables:

Find an n by n symmetric matrix $X = \{x_{ij}\}$ such that:

1. X is positive semidefinite.
2. All constraints (1) are satisfied.

This is not a linear program, due to the constraint that X is psd. This is a convex constraint, or equivalently, a collection of infinitely many linear constraints, but is not a linear constraint by itself. The feasible region is some convex body, but not necessarily a polytope. Nevertheless, the Ellipsoid algorithm can be used to solve semidefinite programs. Unlike the case for linear programming, the algorithm does not return an exact solution, but rather an arbitrarily good approximation to the solution. (The reason for this difference is that it is not necessarily true that a feasible or optimal solution to an SDP is rational with small denominator, or even rational at all.)

To run the ellipsoid algorithm we need certain conditions to hold.

1. Have a bounding ball that contains all feasible solutions (if any exist). In our case, assuming without loss of generality that one of the embedded points is at the origin, no x_{ij} needs to be larger than $\max[(d_{ij})^2]$, and this provides the bounding ball.
2. If the SDP is feasible, we need the feasible region to contain a ball of some small radius ϵ. Here this is done by relaxing the linear constraints to $(d_{ij}^-)^2 - 2\epsilon \leq x_{ii} - 2x_{ij} + x_{jj} \leq (d_{ij}^+)^2 + 2\epsilon$. We do not add slackness to the symmetry of X, in the sense that we keep $x_{ij} = x_{ji}$ as the same variable. To relax the constraint that X is psd, we may require that all its eigenvalues are at least $-\epsilon$ rather than nonnegative.
3. Be able to test if a point (a matrix X) is feasible for the relaxed SDP. Testing the linear constraints and symmetry of X is easy. To test the relaxation for X being psd, compute the eigenvalues of X and test that they are at least $-\epsilon$.

4. Find a separating hyperplane of X in not feasible. For a violated linear constraint, this follows as in linear programming. If the psd constraint is violated, let v be an eigenvector that corresponds to a negative eigenvalue that witnesses it. Then the constraint $v^t X v \geq -\epsilon$ is linear in $\{x_{ij}\}$, violated by the current solution, and holds in every feasible solution (of the relaxed SDP).

Hence the ellipsoid algorithm is applicable to our SDP (and to semidefinite programming in general). It is stopped when the size of ellipsoids become sufficiently small so that they do not contain a ball of radius ϵ (when one of the eigenvalues of the matrix B that represents the ellipsoid drops below ϵ^2). At this point we either have a solution for the relaxed SDP, or the relaxed SDP is not feasible (in which case we abort).

Given a solution to the relaxed SDP, we need to decompose the matrix X to QQ^t and then the rows of Q serve as the desired embedding. This can be done if X is positive semidefinite. However, a minor problem that might arise is that due to the fact that in the relaxed SDP the matrix X is not psd but only very close to being so. It might have some eigenvalues that are negative but very close to 0. To overcome this problem, add 2ϵ to all entries along the diagonal of X, raising all eigenvalues by 2ϵ. Now the matrix becomes positive definite, with very little affect on the constraints. Namely, $(d_{ij}^-)^2 - 2\epsilon \leq x_{ii} - 2x_{ij} + x_{jj} \leq (d_{ij}^+)^2 + 2\epsilon$ changes to $(d_{ij}^-)^2 \leq x_{ii} - 2x_{ij} + x_{jj} \leq (d_{ij}^+)^2 + 4\epsilon$.

Hence if an embedding that satisfies all constraints exist, the ellipsoid algorithm can be used in order to find an embedding that satisfies all constraints up to some small additive error of $O(\epsilon)$. The running time is polynomial in the input and in $\log \frac{1}{\epsilon}$.

We remark that the whole discussion in this section assumed that the dimension of the host space for the embedding is allowed to be arbitrarily large. If one restricts the dimension, the problems studied in this section become NP-hard (even to approximate).