ORIE 6300 Mathematical Programming I November 6, 2008

Lecture 19

Lecturer: David P. Williamson *Scribe: Anna Blasiak*

1 Determining the input and output size of an LP

Recall from last time that an algorithm is good (efficient, polynomial-time) if the number of steps can be bounded by a polynomial in the input size. We asked the question whether LP has a polynomial-time algorithm. We recall that the simplex method has no known pivoting rule that leads to a good algorithm. Today we will start considering alternative algorithms for LP that do run in polynomial time.

We will consider the following linear program:

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

Assume A, b, c are integer.

What is the input size of this linear program? Encoding an integer n in binary takes $\lceil \log_2 (n+1) \rceil + 1 \equiv \text{size}(n)$ bits. Encoding an vector v in binary takes $\sum_{j=1}^{n} \text{size}\,(v_j)$ bits $\equiv \text{size}(v)$ bits. Encoding an matrix A in binary takes $\sum_{j=1}^{n} \sum_{i=1}^{m} \text{size}\,(a_{ij})$ bits $\equiv \text{size}(A)$ bits. We then say that the size of LP input is $L = \text{size}(A) + \text{size}(b) + \text{size}(c)$, $L \geq mn$.

What is the size of the output? We need to be able to bound this by a polynomial to have any hope of achieving a polynomial-time algorithm, since otherwise we will not be able to write down the output in polynomial time. Let U be the maximum size of a_{ij}, b_i, c_j, so that $L = O(mnU)$.

To determine the size of the output, we ask what is the size of writing down a basic solution? We know that a basic solution is the solution to some linear system $\overline{A}x = \overline{b}$, for $\overline{A}$ and $\overline{b}$ part of the input A and b. Then by Cramer's rule,

$$x_j = \frac{\det(\overline{A}_j)}{\det(\overline{A})},$$

where $\overline{A}_j$ is $\overline{A}$ with the j^th column replaced by b. Recall that

$$\det(\overline{A}) = \sum_{\sigma \in S_n} (-1)^{sgn(\sigma)} a_{1\sigma(1)} a_{2\sigma(2)} ... a_{n\sigma(n)},$$

where S_n is the set of permutations of $\{1, 2, ..., n\}$. Since each $a_{i\sigma(i)}$ is at most U bits, the product is $\leq nU$ bits.

Adding together two m-bit numbers gives an $(m+1)$-bit number. We can add together the $n!$ numbers that are at most nU bits in a binary fashion. The addition tree has depth $\log(n!)$ and at every level the number of bits needed to represent the number increases by one. So, $\det(\overline{A})$ is expressible with

$$nU + \log(n!) \leq nU + \log(n^n) = nU + n \log(n)$$

bits. Therefore, since we have m nonzero x_j in the output, the output size is $O(m(nU + n \log(n))$ bits, which is polynomial in the input size.

2 The Ellipsoid Method for LP

We now turn to the first known polynomial-time algorithm for solving linear programs. Actually, what the ellipsoid method does is given some bounded polyhedron $P = \{x \in \mathbb{R}^n : Cx \leq D\}$, either finds $x \in P$ or states that $P = \emptyset$; that is, P is infeasible. How can we use this feasibility detector to solve an optimization problem such as $\min c^T x : Ax \leq b, x \geq 0$? We claim that we can do this by making three calls to the ellipsoid method.

Step 1: Check for primal feasibility: $\exists\ x$ s.t. $Ax \leq b, x \geq 0$? If not, we're done, since the LP is infeasible.

Step 2: Check dual feasibility: $\exists\ y$ s.t. $-A^T y \leq c, y \geq 0$? If not, the primal is unbounded, so we're done.

Step 3: Find optimal solution: $\exists\ (x, y)$ s.t. $Ax \leq b, -A^T y \leq c, x \geq 0, y \geq 0, -b^T y = c^T x$?

By our proof of strong duality, we know that if both the primal and dual are feasible, then there will exist feasible solutions to the primal and dual such that their objective functions are equal. Thus if we make it to Step 3, there is a feasible solution, and the algorithm should return optimal (x, y).

2.1 Idea of Ellipsoid method

Let's now give the basic idea of how the ellipsoid method will work.

- Start with a sphere large enough to contain all feasible points. Call the sphere E_0, center a_0.
- If $a_k \in P$, done. Return a_k.
- If not, $a_k \notin P$, then $C_j a_k > d_j$ for some j.
  - Divide ellipsoid in half with a hyperplane parallel to the constraint $C_j x = d_j$.
- Compute new ellipsoid E_{k+1}, center a_{k+1}, containing the half of E_k that contains P. Repeat.

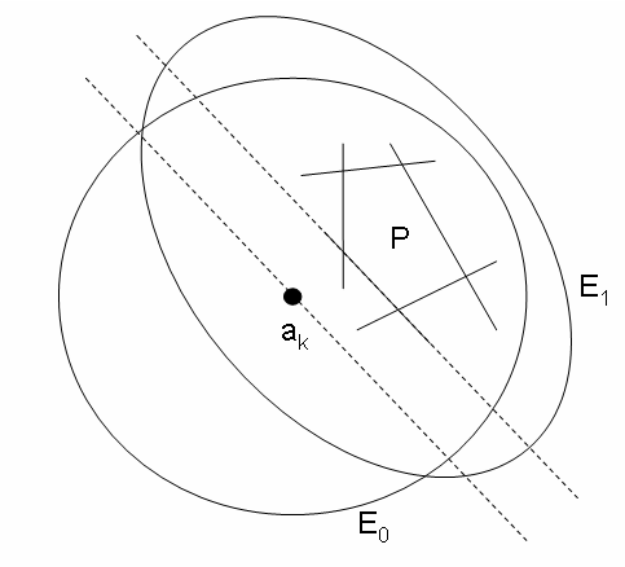
2.2 Issues

This suggested algorithm points to a number of issues that we'll have to resolve in order to have a working algorithm.

- Size of starting Ellipsoid. How big does the initial sphere need to be to contain the whole feasible region?
- Progress. How do we know that we're making some progress towards a solution in each iteration of the algorithm?
- Termination. If P is the empty set, how do we know?

We can start to address some of these issues. For the size of starting ellipsoid: Our initial ellipsoid will be a sphere centered at origin. We know for any vertex x, $|x_j| \leq 2^{nU + n \log n}$, so sphere of radius 2^L, volume $2^{O(nL)}$, will contain the feasible region.

In order to show progress, we will show that after any $O(n)$ iterations, the volume of the ellipsoid will be reduced by a factor of ≈ 2. To show termination, we will show that if P is feasible, then it has a region of volume $2^{\Omega(nL)}$.



Note that these two claims together will imply that the algorithm runs in polynomial time: After $O(n^2L)$ iterations (n per factor of 2, $O(nL)$ factors of 2), either we find a feasible point or the ellipsoid has volume smaller than any feasible region, so P is infeasible.

2.3 Initial ideas

As a start, consider E_0 centered at the origin, radius 1. Consider dividing E_0 with plane $x_1 \geq 0$. Thus

$$E_0 = \{x \in \mathbb{R}^n : \sum_{i=1}^{n} x_i^2 \leq 1\}.$$

We now consider the ellipsoid

$$E_1 = \left\{ x \in \mathbb{R}^n : \left(\frac{n+1}{n}\right)^2 \left(x_1 - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2} \sum_{i=2}^{n} {x_i}^2 \leq 1 \right\}.$$

We'll show next time that E_1 contains all points in the good part of E_0, and the volume of E_1 is relatively smaller than E_0:

$$\text{Vol}(E_1) \leq e^{-\frac{1}{2(n+1)}} \text{Vol}(E_0).$$

Note that if this inequality holds true for all subsequent iterations, then after every $2(n+1)$ iterations, the volume of the ellipsoid will have dropped by a factor of $e > 2$, as claimed.