18.415/6.854 Advanced Algorithms	October 1, 2001
Lecture 6	
Lecturer: Michel X. Goemans	*Scribe: Bin Song and Hanson Zhou*

1 Issues with the Ellipsoid Algorithm

As shown in the last lecture, the update formulae contain a square root in $b = \frac{1}{\sqrt{c^T A_k c}} A_k c$. This is problematic as we do not have infinite bits to represent the result of the square root. Without going into detail, this can be solved by rounding to a new center, taking a slightly bigger ellipsoid, and showing that the new ellipsoid is still small enough to satisfy the required ratio of volumes inequality. Another issue is that at each iteration of the algorithm, the number of bits we need to represent our ellipsoid may grow too fast. Again, this may be handled by rounding and perturbing, the details of which we shall forego.

2 Optimization and Separation

So far, we have seen three kinds of linear programming algorithms:

1. Simplex - not known to be polynomial, but works well in practice.
2. Ellipsoid - polynomial, but works poorly in practice as the ratio of volumes bound is very close to a lower bound as well.
3. Interior Point - polynomial, works well in practice.

The Ellipsoid algorithm does have a redeeming quality in that it only requires us to solve a separation problem: given x, either claim that $x \in P$ or output an inequality $c^T y \leq d$ such that $c^T x > d$ (i.e., x does not satisfy this inequality) and $P \subseteq \{x : c^T x \leq d\}$ (i.e., P satisfies the inequality). Therefore, we do not need to list all of the inequalities of the linear program in order to be able to use the ellipsoid method to solve it. This is an important feature of the ellipsoid algorithm, and enables us to solve some exponential size linear programs.

In summary, the Ellipsoid algorithm requires three things:

1. P bounded, $P \subseteq [-2^Q, 2^Q]^n$
2. If P is non-empty, then $x + [-2^{-Q}, 2^{-Q}]^n \subseteq P'$, where is x is a solution of the original (non-inflated) problem, and P' is the inflated polyhedron.
3. Polynomial time algorithm for the separation problem.

Given these, we can find in $\mathcal{O}(n^2 Q)$ calls to the separation algorithm a point in P' or claim that P is empty.

Here is a simple example of the use of such a separation algorithm. Suppose we would like to optimize over the following linear program (where n is even):

$$\min \left\{ \sum_j c_j x_j : x \in P \right\}$$

where

$$P = \left\{ x : \sum_{j \in S} x_j \geq 1 \text{ for all } S \subset \{1, \cdots, n\} \text{ with } |S| = n/2, x_j \geq 0 \text{ for all } j \right\}.$$

There are $\binom{n}{n/2}$ constraints and this is exponential in n. If we are given this linear program implicitly (saying every subset of size $n/2$ should sum to at least 1), the size of the input would be of the order of $size(c)$ (which is $\geq n$), and thus we can't afford to write down explicitly the linear program and then use a polynomial-time algorithm for linear programming (polynomial in the size of the explicit linear program, and not just $size(c)$).

However, we can use the ellipsoid algorithm to find a point in say $P \cap \{x : c^T x \leq \lambda\}$. Indeed to check whether a point (the center of an ellipsoid) is in P, we do not need to explicitly check every inequality. We can simply sort the x_i's, take the $n/2$ smallest, and verify whether they sum to at least 1. If they do, every inequality in the description of P is satisfied, otherwise we take as S the indices of the $n/2$ smallest values of x and use the inequality over S. This requires only $O(n \log n)$ time (we do not even need to use sorting, we can simply use a linear-time algorithm for selecting the $n/2$-th element; this would take $O(n)$ time).

To decide which ellipsoid to start with, we can use the fact that we can assume $x_j \leq 1$ (if one of the c_j's is negative the problem is unbounded and thus there exists an x of cost below the value λ; otherwise any value greater than 1 can be reduced to 1 without increasing the cost or losing feasibility). To transform P into P' to guarantee the existence of a small ball within it, we can consider vertices of $P \cap \{x : c^T x \leq \lambda\}$. Using Cramer's rule as usual, we can express any such vertex as $(\frac{p_1}{q}, \frac{p_2}{q}, \cdots, \frac{p_n}{q})$ where the p_i's and q are bounded by 2^Q where $Q = O(n \log n + c_{max})$. (Indeed, the matrices involved in the calculations will be square matrices of dimensions at most $n \times n$ and all entries will be 0 or 1 except in one of the rows in which they will be c_j's, so such determinants will be bounded by $(\sqrt{n})^n n c_{max}$.) Thus instead of P, we consider P' in which the inequalities $\sum_{j \in S} x_j \geq 1$ are replaced by $\sum_{j \in S} x_j \geq 1 - \frac{1}{2^Q}$, and instead of intersecting it with the inequality $c^T x \leq \lambda$, we intersect it with $c^T x \leq \lambda + \frac{1}{2^Q}$. Now, we can use the ellipsoid algorithm to decide if there is a point in $P' \cap \{x : c^T x \leq \lambda + \frac{1}{2^Q}\}$ using $O(n^2 Q) = O(n^3 \log n + n c_{max})$ iterations (each involving a rank-1 update and a call to the separation routine).

A simple $O(n \log n)$ time algorithm to solve this linear program is left as an exercise (the solution given in lecture was incorrect).

3 Interior Point algorithms

Karmarkar '84 gave an interior point algorithm, which stays inside the polytope and converges to an optimal point on the boundary. It considers the problem:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b, x > 0 \ . \end{array}$$

Note that the inequality is strict. To enforce this, we add a logarithmic barrier function to the objective function:

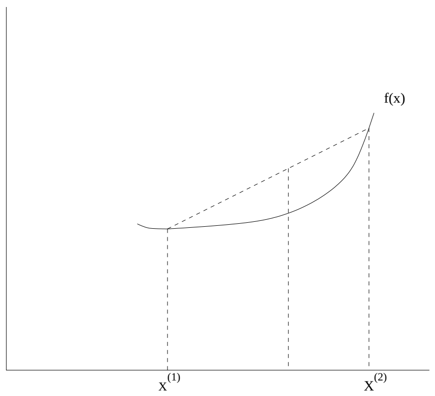

Figure 1: A strictly convex function

$$c^T x - \mu \sum_j \ln(x_j), \quad \mu > 0 \ .$$

Note that this approaches ∞ as $x_j \to 0$. Effectively, this enforces $x > 0$ by imposing an infinite penalty whenever an x component is 0.

Definition 1 *A function f is said to be* ***strictly convex*** *if its domain is convex and for every $x^{(1)} \neq x^{(2)}$ in the domain of f, and every $0 < \lambda < 1$,*

$$f(\lambda x^{(1)} + (1-\lambda)x^{(2)}) < \lambda f(x^{(1)}) + (1-\lambda)f(x^{(2)}).$$

Figure 1 illustrates a strictly convex function.

Lemma 1 *$c^T x - \mu \sum_j \ln(x_j), \mu > 0$ is strictly convex over $x > 0$.*

Proof of lemma 1: This is more or less clear from the graph of the logarithmic barrier function. $\square$

Lemma 2 *If the barrier problem $BP(\mu)$ is feasible and its value finite then there exists a unique solution to it: Minimize $c^T x - \mu \sum_j \ln(x_j), \mu > 0$ s.t. $Ax = b, x > 0$*

Proof of lemma 2: Assume $\exists x^{(1)} \neq x^{(2)}, f(x^{(1)}) = f(x^{(2)}) = BP(\mu)$. Then

$$\lambda x^{(1)} + (1-\lambda)x^{(2)}, 0 < \lambda < 1$$

is feasible since $\{x : Ax = b, x > 0\}$ is convex, and

$$f(\lambda x^{(1)} + (1-\lambda)x^{(2)}) < \lambda f(x^{(1)}) + (1-\lambda)f(x^{(2)}) = BP(\mu),$$

which is in contradiction with the minimality of $BP(\mu)$. Therefore, the solution is unique. $\square$

3.1 Optimality Condition

Lemma 3 *For any $\mu > 0$, x is optimum for $BP(\mu)$, if and only if, $\exists y, s$, such that*

$$Ax = b, \quad x > 0 \tag{1}$$

$$A^T y + s = c \tag{2}$$

$$x_j s_j = \mu, \quad \forall j. \tag{3}$$

Proof of lemma 3: Let $f(x) = c^T x + \mu \sum_j \ln(x_j)$.

By Lemma 1, $f(x)$ is strictly convex, thus a local minimum is also global minimum. By Lemma 2, such minimum is unique.

Under the constraint $Ax = b$, x is optimum for $f(x)$, if and only if, $\nabla f(x)$ is orthogonal to the constraint space $\{x : Ax = 0\}$. Since the column space of A^T is orthogonal to the null-space of A, $\nabla f(x)$ must be in the column space of A^T. Thus, x is optimum, if and only if, $\exists y$, such that $\nabla f(x) = A^T y$.

Since

$$\frac{\partial f}{\partial x_j} = c_j - \frac{\mu}{x_j},$$

we obtain that, x is optimum, if and only if, $\exists y$, such that, $c - \mu X^{-1} e = A^T y$, where

$$X = \begin{pmatrix} x_1 & 0 & 0 & \dots & 0 \\ 0 & x_2 & 0 & \dots & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & 0 & \dots & x_{n-1} & 0 \\ 0 & 0 & \dots & 0 & x_n \end{pmatrix},$$

and e is the n-dimensional column vector with each entry equal to 1.

Let $s_j = \mu / x_j$, we conclude out proof. $\square$

Equations 1, 2, and 3 are called Karush-Kuhn-Tucker optimality conditions.

Note that, since $x > 0$ and $\mu > 0$, we must have $s > 0$. Equation 2 represents the condition of the dual problem of our original problem. The duality gap is

$$c^T x - b^T y = x^T s = \sum_j x_j s_j = \mu n. \tag{4}$$

As $\mu \to 0$, the duality gap tends to 0, and thus $c^T x$ and $b^T y$ tend to the optimal value of (P) and (D).

For a given μ, it is not easy to solve the equations 1, 2, and 3, since equation 3 is not linear. Instead, we will use an iterative method to find the solutions. We start at some μ_0. For each step i, we will go from μ_i to some $\mu_{i+1} < \mu_i$. We show that if we are close enough to μ_i before the step, then after the step, we will be close enough to μ_{i+1}.

The iterative step is as follows. Suppose we are at (x, y, s), where

$$\begin{aligned} x : &\quad Ax = b, \quad x > 0 \\ (y, s) : &\quad A^T y + s = c, \quad s > 0 \end{aligned}$$

We replace x by $x + \Delta x$, y by $y + \Delta y$, and s by $s + \Delta s$. We get

$$\begin{array}{rcl} A(x + \Delta x) = b & \Leftrightarrow & A\Delta x = 0 \\ A^T(y + \Delta y) + (s + \Delta s) = c & \Leftrightarrow & A^T \Delta y + \Delta s = 0 \\ (x_j + \Delta x_j)(s_j + \Delta s_j) = \mu & \stackrel{\text{linearize}}{\longrightarrow} & \Delta x_j s_j + \Delta s_j x_j = \mu - x_j s_j \ , \quad \forall j. \end{array}$$

The equations we obtained above are all linear, and can be solved in polynomial time. Then we will move to the next step with the new x, y, s. We will show that with properly chosen μ_i, x converges to the optimal solution in polynomial time.