CS 224: Advanced Algorithms Fall 2014

Lecture 16 — October 23, 2014

Prof. Jelani Nelson *Scribe: Colin Lu*

1 Overview

In the last lecture we explored the simplex algorithm for solving linear programs. While it runs quickly for many practical applications, it actually does not run in polynomial time.

In this lecture we will finish analyzing the simplex algorithm, and then look into strong duality and complementary slackness. We will then overview a polynomial time algorithm for solving linear programs, the ellipsoid algorithm. We will also introduce the Path-following interior point algoirhtm for solving linear programs, which is polynomial time, and faster in practice than the ellipsoid algorithm.

2 Simplex stuff

As before, we wish to use the simplex algorithm to solve the linear program: $\min c^T x | Ax = B, x \geq 0$.

The simplex algorithm moves from vertex to vertex of the polytope $P = \{x : Ax - b, x \geq 0\}$. A vertex $x \in P$ is a point such that A_x has full column rank, where A_x is a submatrix of A specified by indices $B'_x = \{j : x_j > 0\}$. Since the row rank is at most m, we have that $|B'_x| \leq m$.

For each matrix x, assign basis $B_x \subset \{1, ..., n\}$ with $|B_x| = m$ such that A_{B_x} has rank exactly m. We can think of B_x as B'_x with additional columns so that it has full rank.

In the simplex algorithm, we start at some vertex $x^{(0)}$ (we explored the process of selecting this vertex in the previous lecture), and greedily move to subsequent vertices until some halting condition.

At any time, we represent our current vertex by some basis $B \subset \{1, ..., n\}$, $|B| = m$, and let $N = [n] \backslash B$.

We can think of the linear program as $\min c_B^T x_B + C_N^T x_N$ subject to $A_B x_B + A_N x_N = b$, with $x_B, x_N \geq 0$.

By muliplying both sides, we get that this is the same as minimizing $c_B^T A_B^{-1} b + (c_N - A_N^T (A_B^{-1})^T c_B)^T x_N$.

We define $\tilde{c}_N$ as $(c_N - A_N^T (A_B^{-1})^T c_B)^T$, the "reduced cost"

The simplex algorithm proceeds as follows:

1. At first pass, assume $v_B > 0$. (v is the current vertex).
2. While $\exists j \in N$ such that $\tilde{c}_j < 0$, we increase the jth coordinate. This forces a decrease in some coordinates in v, according to the equation $v_B = A_B^{-1}(b - A_N v_N)$. We decrease the jth coordinate until some coordinate of v is 0.

3. Otherwise, if $\tilde{c}_j \geq 0$ for all j, then halt and declare that our current solution is optimal.

2.1 Pivoting Rules

However, in reality, we can't just assume that $v_B > 0$. If $v_B > 0$ doesn't hold, and there exists some index $j \in N$ such that $\tilde{c}_j < 0$, we will add j to B and kick out k from B. However, there will often be a lot of choices for indices j to add, and indices k to remove.

Multiple "pivoting" rules exist for choosing j and k exist that provably avoid infinite loops, although some rules that can result in infinite loops are used in practice occasionally.

One such rule that provably terminates is Bland's rule:

1. Pick j to minimize $\tilde{c}_j$. If multiple such j exist, then just take the smallest such index j.
2. Define $r \in R^n$, $r_B = 0$, $r_N = \tilde{c}_N$. $A' = A_B^{-1}A$, $b' = A_B^{-1}b$, and $q = \min_k\{\frac{b'_k}{A'_{k,j}} | A'_{k.j} > 0\}$
3. Pick k to minimize q. If there is a tie, pick the smallest index k minimizing q.

Bland's rule can be shown to never result in an infinite loop. See the original paper [1] for a proof, or the lecture notes [8].

2.2 Run-time

We will now analyze the running time of the simplex algorithm. At each iteration of the simplex algorithm, we take polynomial time to decrease the jth coordinate to perform a pivot.

The number of iterations is bounded above by the number of vertices, which is at most $\binom{n}{m}$ (since we can specify a vertex by its basis elements). In fact, it can be shown that the number of vertices is at most $\binom{n-\frac{m}{2}}{\frac{m}{2}}$. This follows from the "Upper Bound Theorem" proven by McMullen in[2].

If we wanted to show a better bound than simply $\#vertices \times iterationtime$, one necessary step would be to show that the maximum diameter (longest shortest path) of a graph corresponding to a polytope in d dimensions defined by m hyperplanes is polynomial in m and d . If we were unable to do this, then in the worst case we might simply start "too far" away from the optimal vertex.

In 1957, Warren Hirsch conjectured that $diam \leq m - d$. His conjecture remained open for more than 50 years, before Santos disproved it in [3] with an explicit construction in which $d = 43$, $m = 86$, $diam \geq 44$. He also showed that for fixed d, $\epsilon > 0$, there exists an infinite family of polytopes with $m \to \infty$ that have diameter at least $(1 + \epsilon)m$.

However, even if we managed to show that the maximum diameter of a polytope graph is small, we would also need to show that the simplex algorithm actually finds short paths within such graphs. It's still an open question as to whether there is a fast (i.e. worst case polynomial time) implementation of the simplex algorithm.

2.2.1 Simplex's speed in practice

Spielman and Teng in [4] used "smoothed analysis" to explore why the simplex algorithms we use are fast in practice, despite the fact that haven't shown to be polynomial time in general.

In smoothed analysis, we consider the average running time of a worst case instance with some small random independent noise added to all entries. Formally, the running time of an algorithm A in smoothed analysis is $\min_x \mathbb{E}(runtime(A(\tilde{x})))$, where $\tilde{x}$ is x with random noise added, x is some input to the algorithm, and the expectation is taken over the noise added to x.

Spielman and Teng showed that simplex algorithms have smoothed analysis times that are polynomial.

3 Strong Duality

Recall that the dual of our linear program $\min c^T x | Ax = B, x \geq 0$, is the program $\max y^T b | A^T y \leq c$. Note that when we have equality constraints $Ax = b$, the dual variable y does not need to be non-negative.

There are multiple ways to get strong duality. We will show it as a corollary of the simplex algorithm.

Define the "shadow price vector", $y = (A_B^{-1})^T c_B$.

Claim 1. $c^T v_B = b^T y$.

Proof. Recall that we have $A_B v_B = b$, which tells us that $v_B = A_B^{-1} b$. Hence, $c^T v_B = c^T A_B^{-1} b = b^T (A_B^{-1})^T c_B = b^T y$. □

This does not quite give us strong duality: we need to also show that y is a feasible solution to the dual.

Define the dual slack as $r = c - A^T y$. y being feasible is equivalent to $r \geq 0$. The definition of y gives us that $r_B = 0$, and that $r_N = \tilde{c}_N$. Note that the simplex algorithm halts when $\tilde{c}_N \geq 0$, which implies that at this point, we have dual feasibility.

4 Complementary Slackness

We can define the "complementarity gap" as $x^T r = x^T (c - A^T y) = c^T x - (Ax)^T y = c^T x - b^T y$.

Theorem 2. *(complementary slackness): If x^* and y^* are optimal for the primal and dual respectively, then for all i, either $x_i = 0$, or $r_i = (c - A^T y)_i = 0$.*

Proof. At optimality, by strong duality, there is no gap between the primal and dual solutions, so $c^T x - b^T y = 0 = \sum x_i r_i$. Since each x_i, r_i is non-negative, this can only be true if for each i, either $x_i = 0$ or $r_i = 0$, as required. □

5 Ellipsoid Algorithm

The first polynomial-time algorithm for solving linear programs is the ellipsoid algorithm, which was introduced by Shor, and applied to linear programs by Khachian in [6]. We won't cover the full details of ellipsoid here but will just give a flavor of what goes on. If you want to see more details, see for example the notes from lectures 4 through 6 of [5].

The ellipsoid algorithm itself tests the feasibility of some polytope $P = \{x : Ax \leq b\}$.

The ellipsoid algorithm only uses the constraints in one way: it wants an oracle that will tell it, given an infeasible point, a constraint that it violates. However, if an oracle is given indepednent of the contraints themselves, then that can be used instead of directly referencing them.

Given some $a \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}$, the ellipsoid $E(a, A) = \{x | (x-a)^T A^{-1}(x-a) \leq 1\}$.

The ellipsoid algorithm is used to test the feasibility of some polytope $P = \{x : Ax \leq b\}$.

The basic idea of the algorithm is to:

- Start with some ellipsoid E_0 which is guaranteed to contain P. One way to do this is to compute the bit complexity of P, and use this to put an upper bound on the radius of a sphere needed to contain it.
- Let a be the center of the current ellipsoid E. While a is not in P, there exists some hyperplane H corresponding to some constraint, such that P lies entirely on the opposite side of H from a. Create a new ellipsoid E' which still contains P and has smaller volume and repeat. Khachian showed that there is a way to do this such that $\frac{\text{vol}(E')}{\text{vol}(E)} \leq e^{\frac{-1}{2n}}$.
- If $a \in P$, return a. If $\text{vol}(E)$ has become "too small", declare that P is empty.

This has a problem, in that the polytope P could lie in a lower dimensional subspace, such that it has zero volume, but still contains points. To fix this, we do some pre-processing:

Theorem: Let L be the "bit complexity" of P. $L \sim \log(\max_{i,j} |A_{ij}| + \|b\|_\infty + m + n)$. Define $P' = \{x : Ax \leq b + 2^{-L}, \forall j, -2^L \leq x_j \leq 2^L\}$. Then P is infeasible $\iff P'$is infeasible. Thus, we can then apply the above algorithm to P' in order

5.1 Application to solving linear program

Suppose we actually want to solve a linear program $\min c^T x$, such that $Ax = b$, $x \geq 0$.

We can solve it using the ellipsoid algoirthm as follows:

1. Check if $P = \{x | Ax = b\}$ is feasible. If not, output "not feasible".
2. Write the dual: $\max b^T y$ such that $A^T y \leq c$. Check that $P_{dual} = \{y : A^T y \leq c\}$ is feasible. If not, then output "unbounded".
3. Find (x, y) that is feasible for the polytope defined by $\{Ax = b, x \geq 0, A^T y \leq c. c^T x = b^T y\}$. This pair (x, y) give optimal solutions for the primal and dual respectively.

5.1.1 Run-time of ellipsoid algorithm on linear programs

For the kth ellipsoid E_k, we have that $\text{vol}(E_k) \leq e^{\frac{-k}{2n}}\text{vol}(E_0)$. We stop the algorithm when this volume is less than that of P', which takes a polynomial number of iterations.

6 Path-following interior point

Path-following interior point is another algorithm that gives polynomial time solutions to linear programs. It was introduced by Karmarkar in [7].

The interior point algorithm gradually tries to reach an optimal vertex while staying in the interior of P. It introduces a variable that makes it very easy to satisfy the constraints, but is penalized heavily in the objective function. With this variable, it is very easy to find a starting interior point, and then iterates on this point.

For the interior point algorithm, we'll look at linear programs of the form $\min c^T x | Ax \geq b$, and define the slackness $S(x) = Ax - b$. For interior point, we attempt to compute $\min_{x \in \mathbb{R}^n} \lambda c^T x + p(S(x))$ where p is a "barrier function" such that $p(z) \to \infty$ if $z_i \to 0$ for any i. In particular, we'll use the "log barrier" function, $p(z) = -\sum_{i=1}^{m} \ln(z_i)$.

For very small λ, $\min_{x \in \mathbb{R}^n} \lambda c^T x + p(S(x))$ is a kinda of central point in the polytope, and we iterate, incrementing λ and modifying x, until $\min_{x \in \mathbb{R}^n} \lambda c^T x + p(S(x))$ is approximately a vertex, and can then be "rounded" to the nearest vertex. We'll analyze the interior point algorithm in more detail in the next lecture.

References

[1] Robert Bland. New Finite Pivoting Rules for the Simplex Method. *Mathematics of Operations Research*, 2(2):103–107, 1977.

[2] Peter McMullen. The maximum numbers of faces of a a convex polytope. Mathematika 17:179-184, 1971.

[3] Francisco Santos. A counterexample to the Hirsch conjecture, Annals of Mathematics 176 (10: 383-412, doi:10.4007/annals.2012.176.1.7, 2011.

[4] Daniel A. Spielman, Shang-Hua Teng. Smoothed analysis of algorithms. Why the simplex algorithm usually takes polynomial time. J. ACM 51(3): 385-463, 2004.

[5] Michel Goemans. Combinatorial Optimization Lecture notes on the ellipsoid algorithm. `http://www-math.mit.edu/~goemans/18433S09/ellipsoid.pdf`.

[6] Leonid Khachian. A polynomial algorithm in linear programming. Doklady Akademi Nauk USSR 244:1093-1096, 1979.

[7] Narendra Karmarkar. A New Polynomial Time Algorithm for Linear Programming. Combinatorica, Vol 4, 4:373-395, 1984.

[8] Yinyu Ye. MS&E310 Linear Optimization Autumn Course Notes on Bland's Rule. http://web.stanford.edu/class/msande310/blandrule.pdf.