CS 270: Combinatorial Algorithms and Data Structures Spring 2023

Lecture 22 — April 6, 2023

Prof. Jelani Nelson *Scribe: Sudhanva Kulkarni*

1 Overview

In the last lecture we looked at simplex for a given a linear program

$$\begin{aligned} \min c^T x \\ Ax = b \\ x \geq 0 \\ A \in \mathbb{R}^{m \times n} \end{aligned}$$

Recall that a vertex is determined by a basis $B \subseteq [n]$ with $|B| = m$. Also $v_B = A_B^{-1} b$.
In this lecture we cover

1. Strong Duality
2. Complementary Slackness
3. Ellipsoid algorithm
4. Internal point methods (didn't quite get to this)

2 Simplex and Strong Duality

Theorem 2.1 (Strong Duality)**.** *If the primal that is bounded and feasible, then its dual is also bounded and feasible. Further, they have the same optimal value.*

We'll prove the above by looking at the termination condition for simplex. Recall from last lecture that if we fix a basis B, we may rewrite our linear program as follows

$$\begin{aligned} \min c_B^T x_B + c_N^T x_N \\ A_B x_B + A_N x_N = b \\ x_N, x_B \geq 0 \end{aligned}$$

Where x_B is just all components with index belonging to B and x_N is the rest. Then, using the equality constraint, we may write

$$x_B = A_B^{-1} b - A_B^{-1} A_N x_N$$

and thus rewrite our objective as

$$(c_N - A_N^T (A_B^{-1})^T c_B)^T x_N = \tilde{c_N} x_N$$

Simplex terminates when $\tilde{c_N} \geq 0$.

2.1 Proof of Strong Duality

Now, to prove strong duality, say we run simplex and achieve $c_N \geq 0$ with a basis B. Note that $x_B = A_B^{-1} b$. Now, for strong duality, we'd want $b^T y = c_B^T x_B$ where y is dual feasible. Note that $b^T y = y^T b = (c_B)^T A_B^{-1} b$. Before we check that y is dual feasible let's write down the dual. We get

$$\max b^T y$$
$$A^T y \leq c$$

Consider the slack vector $s = c - A^T y$. Then

$$s_B = c_B - A_B^T (A_B^{-1})^T c_B = c_B - A_B^T (A_B^T)^{-1} c_B = 0$$

and

$$s_N = c_N - A_N^T (A_B^{-1})^T c_B = \tilde{c_N}$$

Since simplex terminated, $c_n \geq 0 \implies s \geq 0$ and so y is feasible. Hence we have strong duality.

2.2 Runtime of Simplex

Since simplex essentially looks at each vertex of the constructed polytope, we can try to get a bound on its runtime by looking at maximum distance we'd have to travel in said polytope. To do this, construct a graph corresponding to the polytope in the obvious way. Hirsch conjectured in '57 that that the diameter of such a graph is bounded by $m - n$. This conjecture was disproved in 2011 by Santos [2] who showed that $\exists P \subseteq \mathbb{R}^{43}$ which is an intersection of 86 half spaces but its diameter is ≥ 44.
While this was certainly disappointing, simplex still worked well in most practical cases, especially with appropriate pivoting. In 2003, Spielman and Teng [1]showed that there exists a pivoting rule P such that $\forall I = (A, b, c)$, we have that $\mathbb{E}(runtime(Perturb(I)))$ is polynomial. Here $Perturb(I)$ means that we add some small amount of noise to each of the variables in our program (say Gaussian with mean 0 and extremely small variance).

3 Complementary Slackness

Theorem 3.1 (Complementary Slackness)**.** *Say the primal and dual are both bounded and feasible with optimal solutions x, y respectively. Then, if $s = c - A^T y$, we have $\forall i \in [n]$*

- $x_i > 0 \implies s_i = 0$
- $s_i > 0 \implies x_i = 0$

3.1 Proof

The proof of the above uses Strong Duality. By Theorem 2.1 we have

$$c^T x - b^T y = 0 \tag{1}$$

Now let's consider $x \cdot s = x^T s$. We have

$$x^T s = x^T(c - A^T y) = c^T x - (Ax)^T y = c^T x - b^T y$$

So, we get

$$\sum_i x_i s_i = c^T x - b^T y \tag{2}$$

from (1) and (2) we get $\sum_i x_i s_i = 0$. But since $x \geq 0$ and $s \geq 0$, the only way we can sum to 0 is if each summand is 0. So one term in each summand must be 0.

4 Ellipsoid Algorithm

Ellipsoid was invented by Khachiyan in 1979 [3] , 32 years after simplex. It is (weakly) polynomial in m, n, l which represent the number of variables, constraints and bit complexity. The only requirement for ellipsoid to work is a sub-routine that has the following specification - If we give the subroutine a point as an input, it is able to give us either one constraint that the point violates, or it tells us if the point is feasible. Note that ellipsoid doesn't actually solve for optimum. It only gives us a feasible point (But we'll see how to get around this in a very straightforward way).

4.1 Geometric intuition for algorithm

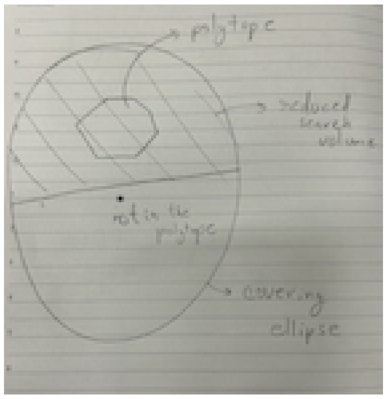
Given any symmetric PSD matrix P, we describe an ellipse with centre α by

$$(x - \alpha)^T P (x - \alpha) \leq 1$$

Or equivalently

$$||B(x - \alpha)||_2^2 \leq 1$$

Where $A = B^T B$. The unit sphere with centre at origin is seen as a special case of this by just letting $A = I$, $\alpha = 0$.
A polytope, as seen before, is described by a system of inequalities $Ax \leq b$.
The idea behind ellipsoid is to keep finding ellipses that cover our polytope, use the "checking" subroutine on some point, and then lower our volume of search accordingly



4.2 The algorithm

Algorithm 4.1. Ellipsoid:

- Set E_0 to be some extremely large sphere containing our polytope P
- for $i = 0, 2, \ldots k$
  1. Run the constraint checking subroutine on the centre of E_i
  2. If the point is feasible, return
  3. Else let E_{i+1} be the smallest ellipsoid containing the intersection of the half-plane returned in step 1 and E_i

Thankfully, the step of constructing the new ellipse is a closed form problem that can be solved easily; Unthankfully, we don't show the solution here.
Showing that this algorithm works relies on the fact that $vol(E_{k+1})/vol(E_k) \leq \exp \frac{1}{2(n+1)}$. Then, since our first elliposid is an n-dimensional sphere, we have

$$vol(E_0) = \frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1)} r^n$$

So after k iterations, we're bounded by something on the order of $(nR)^n \cdot \exp \frac{k}{2(n+1)}$. Picking k appropriately large, we see that the algorithm terminates.

4.3 Modifications to Ellipsoid

As we just saw, Ellipsoid only returns a feasible point, not the optimal one. However, we can change this by simply solving the primal and dual at the same time! Then any feasible solution is optimal. So, we input the constraints

$$\begin{aligned} Ax &= b \\ A^T y &\leq c \\ x &\geq 0 \\ c^T x &= b^T y \end{aligned}$$

Another issue one may run into is the polytope being "too thin", so its volume would be extremely low and so we'd need extremely large k for this process to terminate. We can solve this by using our algorithm on a newly constructed polytope P' which is given by

$$\begin{aligned} P' = \{(x, z) : Ax &\leq b + z \cdot \mathbf{1} \\ \forall i - 2^l &\leq x_i \leq 2^l \\ \forall i - 2^l &\leq z_i \leq 2^l \end{aligned}$$

This beefs up our polytope and makes convergence more realistic.

5 Next Lecture

In the next lecture, we'll go over interior point methods for optimization problems.

References

[1] Daniel A. Spielman and Shang-Hua Teng, (2003) Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time

[2] Francisco Santos (2012) A counterexample to the Hirsch Conjecture, Annals of Mathematics

[3] Khachiyan, Leonid Genrikhovich (1979) A polynomial algorithm in linear programming, *Doklady Akademii Nauk*