COMS 4995-3: Advanced Algorithms April 5, 2017

Lecture 20 – Linear Programming: Simplex and Ellipsoid Algorithms

Instructor: *Alex Andoni* Scribes: *Andrew Aday*

1 Introduction

In today's lecture we go over the intuition behind the Simplex and Ellipsoid algorithms. To recap, recall that we are trying to minimize $c \cdot x$ under the constraint $Ax \geq b$.

2 Simplex [Dantzig, 1949]

The simplex algorithm is actually an entire class of algorithms which follow a certain procedure. Each step in this procedure has many possible algorithmic approaches, and thus the simplex algorithm has many variants. It remains an open question whether there exists a variant which can find the optimal solution for an arbitrary polytope in polynomial time. Some definitions are necessary to explain the algorithm:

Definition 1. *A* **vertex** *is a feasible point (i.e. a point contained within the feasible region) that is defined by n tight, linearly-independent constraints.*

Definition 2. *An* **edge** *is a straight segment connecting 2 vertices that share $n-1$ tight contraints in common. An edge must also be defined by $n-1$ tight constraints.*

Definition 3. *A* **neighbor** *of some vertex x is another vertex y such that $\exists$ an edge of P connecting x and y. Let $N(x) \triangleq \{$neighbors of $x\}$.*

2.1 Algorithm Steps

We assume that the feasible region exists and is bounded.

1. *Select any vertex in the feasible region.* We call this starting point x^0. Note that in the general case, this step is nontrivial. In many real-world problems however, there exists an intuitive starting point that requires no calculation, e.g. a graph with 0 flow for the Min-Max flow problem.

   Note: one way to accomplish this for, say, an LP $Ax \leq b$ is to setup the following LP: $\min t$ where $Ax \leq b + t \cdot \mathbf{1}$.

2. *Move to any neighboring vertex along the direction of $-c$.* More formally, at step t, select a $y \in N(x^{t-1})$ such that $c \cdot y < c \cdot x^{t-1}$. This $x^t = y$ becomes our next point. Note that the greedy solution is **not** optimal for any given polytope (to see why, think of shortest paths in a weighted graph).

3. *Repeat step 2 until we cannot find a neighbor that further minimizes $c \cdot x$.* At this point we have found an optimal solution, and the algorithm terminates. The algorithm must always converge to an optimal solution because the feasible region is both convex and bounded.

To calculate $N(x)$ in step 2, we use a brute-force method. Recall that $x \in \mathbb{R}^m$ is subject to exactly n tight constraints, and $m-n$ "loose" constraints ($\geq$ or $\leq$). Furthermore, all of its neighbors share precisely $n-1$ of these tight constraints. Then to find the neighbors of a point x, we drop one of n possible tight constraints, tighten one of $m-n$ possible "loose" constraints, solve the corresponding linear system, and see if the solution is feasible. More formally, given a starting point x with tight constraints, say, $A_1x = b_1, A_2x = b_2, ..., A_nx = b_n$, we can calculate $\forall j \in \{1, ..., n\}, \forall i \in \{n+1, ..., m\}$ a potential neighbor vertex y with tight constraints $A_1y = b_1, ..., A_{j-1}y = b_{j-1}, A_{j+1}y = b_{j+1}, ..., A_ny = b_n, A_iy = b_i$. After finding y using any method, say Gaussian elimination, we then need to verify $Ay \geq b$ and calculate $c \cdot y$. Step 2 therefore takes time:

$$n(m-n) \cdot [\text{time to do gaussian elimimation} + \underbrace{nm}_{\text{check } Ay \geq b \text{ and } c \cdot y}]$$

The question arises: how do we choose our *pivot*? That is, how do we choose which neighbor to travel to? Again, the greedy choice may not always be optimal.

2.2 Performance

Simplex is used widely in practice. Spielman and Teng have shown that the *smoothed* time complexity of the algorithm is polynomial [1]. But to prove a polynomial running time in all cases, it is necessary to at least prove the following conjecture (which is a major open question):

Conjecture 4 (Hirsch conjecture)**.** *Let G be the graph where each vertex represents a vertex of the feasible region. Two vertexes are connected in G iff they are neighbors in the feasible region. Then for any such G, for all vertices x, y, $\exists$ a path $x \to y$ of length $n^{O(1)}$. I.e. the diameter of G is polynomial.*

3 Ellipsoid Algorithm [Khachiyan 1979]

The ellipsoid algorithm is the first (weakly) polynomial-time algorithm for LP. It really solves the feasibility problem ("is there any feasible point?"). We can use binary search to reduce a general LP to the feasibility problem as follows. We guess a scalar value for the optimal solution $v = c \cdot x^*$, and then see if this guess is *feasible*, i.e. if $\exists x$ such that $Ax \geq b$ and $c \cdot x \leq v$. This the **feasibility problem**. Depending on the feasibility result, we either increase or decrease v, doing binary search until we land at the optimal v^*.

More formally, let $Q_t = P \cap \{x : c^T x \leq t\}$, where P is the feasible region of the original LP problem. The ellipsoid algorithm simply uses binary search to find the smallest possible t such that $Q_t \neq \emptyset$.

Note that the number of feasibility problems we need to solve is polynomial in the bit-size of the input, so to prove an overall poly-time bound we need only show we can solve the feasibility problem in polynomial time.

[1] `http://www.cs.yale.edu/homes/spielman/simplex/`

3.1 Feasibility Problem

Definition 5. *An* **r-ball** *around x is $B_r(x) = \{y : ||y - x|| \leq r\}$*

Definition 6. *An* **axis-aligned ellipsoid** *is $E(x) = \{y \in \mathbb{R}^n : \sum_{i=1}^{n}(y_i - x_i)^2/\lambda_i^2 \leq r^2\}$ for $\lambda_i > 0$*

Definition 7. *A* **general ellipsoid** *is $E(x) = \{y \in \mathbb{R}^n : (y - x)^T A^T A(y - x) \leq r^2\}$, where A is a full-rank n-by-n matrix*

One can intuitively think of the feasibility problem as an "oracle" or "certificate" for the ellipsoid algorithm: Given a polytope described by input constraints ($Ax \geq b$ and $c^T x \leq v$), decide if it is empty. We consolidate these input constraints into the matrix D and define the corresponding polytope as $Q = \{x \in \mathbb{R}^n : Dx \geq e\}$, where $e^T = [b_1, b_2, ..., b_m, -v]$. The algorithm is:

1. At step $t = 0$, begin with an ellipsoid $E_0 = B_R(y_0)$ with, $y_0 = \mathbf{0}$ and R sufficiently large so that $Q \subseteq B_R(\mathbf{0})$;

2. At step $t \geq 1$, check if the center y_{t-1} of the ellipsoid E_{t-1} is in Q. If so, return because $y_{t-1} \in Q \implies Q \neq \emptyset$. Else find a constraint that y_{t-1} has violated. This violated constraint can be visualized as a hyperplane $A_i x \geq b_i$. If we consider a parallel hyperplane $A_i x = A_i y_{t-1}$ (i.e., the hyperplane that passes throught the original of the current ellipsoid): this hyperplane partitions the ellipsoid E_{t-1} into two. We then construct a new ellipsoid E_t which is the smallest possible ellipsoid that still encloses the half of E_{t-1} that contains Q (the right part is easy to identify by just checking the sign of $A_i y_{t-1} - b_i$). E_t is "tighter" than E_{t-1} (it contains only half of the latter!). Formally, one can prove that by considering the volume of the ellipsoids: $vol(E_t) \leq vol(E_{t-1})(1 - \frac{1}{2n})$. See proof in, say, `http://www-math.mit.edu/~goemans/18433S09/ellipsoid.pdf`.

3. If repeating step 2 some polynomial number of times still does not yield a $y_t \in Q$, we are confident that Q is empty. Return.

Formally, for the above to work, we need an additional condition:

Claim 8. *If the starting ellipsoid E_0 has volume V, then in $O(n \lg V/\epsilon)$ steps our E_t we will have volume ϵ (while still containing Q).*

In particular, if the feasible region Q contains a ball of volume ϵ, then the algorithm finds a $y \in Q$ in at most $O(n \lg V/\epsilon)$ steps.

A rigorous proof of the claim 8 can be found here: `http://www-math.mit.edu/~goemans/18433S09/ellipsoid.pdf`

We conclude by saying that ellipsoid algorithm, while polynomial-time, is rarely used in practice and is generally slower than simplex on real-world inputs. The ellipsoid algorithm, however, is not limited to linear programming and may be applied to more general convex optimization problems.