COMPSCI 532: Design and Analysis of Algorithms	September 16, 2015
Lecture 7	
Lecturer: Debmalya Panigrahi	*Scribe: Allen Xiao*

1 Overview

In this lecture, we present several algorithms for solving linear programs. We will mostly refer to linear programs in canonical form:

$$\begin{aligned} \min \quad & c^\intercal x \\ \text{s.t.} \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

Following that, we will introduce the idea of *separation oracles*, on which some of algorithms are based.

2 Geometry of Linear Programs

Earlier, we said that the feasible space of linear programs formed a convex polyhedron:

Definition 1. *A **polyhedral set** is the intersection of a finite number of half-spaces in $\mathbb{R}^n$.*

$$P = \{x \in \mathbb{R}^n \mid Ax \leq b, x \geq 0\}$$

A linear program minimizes/maximizes some objective over points in P.

Definition 2. *A **vertex** of a polyhedral set P is $x \in P$ where for some $y \in P$:*

$$x - y \in P \text{ and } x + y \in P \implies y = 0$$

In other words, x is not a convex combination of any other points in P.

As the name would suggest, vertices occur on the geometric "corners" of the polyhedron. Intuitively, our definition says that it's impossible to move forward *and* backward in *any* direction from x. We now tie this concept back to linear programs:

Definition 3. *A **basic feasible solution** $x \in P$ is one where n linearly independent constraints are tight.*

Lemma 1. *Vertices and basic feasible solutions are equivalent.*

Proof. We prove both directions.

1. (Vertex $\implies$ BFS)

   By contrapositive. Suppose $x \in P$ is not a basic feasible solution. Let each a_i' be a tight constraint:

   $$a_i' x = b_i$$

And let the submatrix of tight constraints be A'. Since x is not a basic feasible solution, there are at most $n-1$ linearly indpendent constraints in A'. Since A' is not full rank, its null space $\text{null}(A')$ has dimension at least 1. For any $y \in \text{null}(A')$, its projection onto A' is 0.

$$\text{proj}_{A'}(y) = 0$$

Subsequently:

$$\begin{aligned} A'(x+y) = A'x &= b \\ A'(x-y) = A'x &= b \end{aligned}$$

Now consider some other constraint in this null space (not tight) a_i. If the problem is not under-constrained, there is at least one.

$$a_i x > b_i$$

We can therefore choose a $y \in \text{null}(A'), y \neq 0$ where:

$$\begin{cases} a_i(x+y) = b_i & \text{if } a_i y > 0 \\ a_i(x-y) = b_i & \text{if } a_i y < 0 \end{cases}$$

This is the direction y which fails the vertex definition, so x is not a vertex.

2. (Vertex $\Longleftarrow$ BFS)

   Suppose that $x \in P$ is not a vertex. Then there exists some line in direction y such that $x+y, x-y$ are both in P. For every tight constraint i (by definition):

$$\begin{aligned} a_i x &= b_i \\ a_i(x+y) &\leq b_i \\ a_i(x-y) &\leq b_i \end{aligned}$$

   Therefore $a_i y = 0$. However, the a_i form a rank n matrix by definition of basic feasible solution, and therefore it must be that $y = 0$.

□

Lemma 2. *Any bounded LP in standard form has an optimum at a basic feasible solution.*

Proof. Let x be optimal for P. If x is not a basic feasible solution, there are less than n linearly independent tight constraints. We will move in a direction which increase the number of tight constraints, without decreasing the objective value. As before, let A' be the submatrix of tight constraints. We can find $y \in \text{null}(A')$ where:

$$A'y = 0$$

For sufficiently small $\varepsilon > 0$:

$$x \pm \varepsilon y$$

is also feasible. Moreover:

$$c^\intercal(x \pm \varepsilon y) = c^\intercal x \pm \varepsilon c^\intercal y$$

Optimality of x means that

$$c^\intercal y = 0$$

The feasible space is bounded, so one of these directions ($\pm y$) will be bounded. We move x in that direction until a constraint becomes tight. □

3 Simplex Algorithm (Dantzig 1947)

Simplex is a class of algorithms which solve linear programs by moving from vertex to vertex until an optimal solution is reached. The variables which require tight dual constraints are called *basic* variables (alternatively, *non-basic*). Simplex swaps a basic variable for a non-basic variable in an operation known as a *pivot*. Gaussian elimination can be used to find the new vertex.

1. (*Phase 1*)

   Find an initial basic feasible solution x, with tight constraints $T \subseteq \{a_1, \ldots, a_m\}$. This can be done by solving another LP which has $x = 0$ as an initial BFS.

2. (*Phase 2*)

   Repeatedly perform cost-improving pivots (swapping out elements of T) until x is optimal (no pivot improves the cost). Since the size of T does not change, x is always a basic feasible solution.

3. Output optimal x.

The specific algorithm depends on the *pivoting rule*, which describes the vertex to be explored next. There is no known pivoting rule for which the simplex algorithm is worst-case sub-exponential time, it is quite successful in practice. For more information on hard instances for simplex-style algorithms, look up the Klee-Minty cube.

3.1 Missing Details

Some things we did not discuss in detail:

- The exact form of the LP to find an initial basic feasible solution. It adds slack variables z_j for each constraint, and tries to minimize their sum. By initially setting $z_j = b_j$, $x = 0$ is feasible.
- Degeneracy. Some instances have pivots where the objective value stays the same; this is called *stalling*. A serious (but rare) related problem is *cycling* of pivots, which make the algorithm fail to terminate. However, there are pivoting rules which avoid cycling.
- Simplex implementation. Most implementations maintain a *tableau* (explicitly or implicitly) tracking the current basis of tight constraints, and are able to evaluate the pivoting rule from this representation.

4 Ellipsoid Algorithm (Khachiyan 1980)

The ellipsoid algorithm is a weakly polynomial algorithm which solves for LP feasiblity, due to Khachiyan [Kha80]. However, LP optimality and LP feasibility are equivalent, so this still "solves" the linear program.

Fact 3. *LP optimality is equivalent to LP feasibility.*

Proof. Consider a linear program:

$$\begin{aligned} \min \quad & c^\intercal x \\ \text{s.t.} \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

The problem of optimality is to minimize the objective:

$$\min c^\intercal x$$

Recall that the dual program gave a tight bound on the objective, by strong duality:

$$\begin{aligned} \max \quad & b^\intercal y \\ \text{s.t.} \quad & A^\intercal y \leq c \\ & y \geq 0 \end{aligned}$$

$$\min c^\intercal x = \max b^\intercal y$$

Then, we can combine these constraints to make a program where only the optimal solutions are feasible.

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

The solution (x^*, y^*), must have:

$$c^\intercal x^* = b^\intercal y^*$$

□

The basic idea for the ellipsoid algorithm is:

1. Maintain an ellipsoid containing the polyhedron P.
2. Check if the center of the ellipsoid is inside P. If so, done.
3. If not, find a separating hyperplane, parallel to the violated constraint and through the ellipsoid center, and split the ellipsoid in half.
4. Enclose the half-ellipsoid containing P in a minimal ellipsoid containing it. Recurse on this smaller ellipsoid.

4.1 Ellipsoid Proof Sketch

We can begin with some origin-centered ball that is sufficiently large. Let E_k be the kth ellipsoid. It is possible to prove that:

$$\frac{\text{Vol}(E_{k+1})}{\text{Vol}(E_k)} < \exp\left(-\frac{O(1)}{n}\right)$$

Additionally,

$$\begin{aligned} \text{Vol}(E_0) &= 2^{O(L)} \\ \text{Vol}(P) &= 2^{-\Omega(L)} \end{aligned}$$

where

$$L = m + n + \log(\det_{\max}) + \log(b_{\max}) + \log(c_{\max})$$

and

$$\det = \max\{\det(A') \mid A' \text{ is a square submatrix of } A\}$$

Finally, the number of iterations is:

$$O\left(n \log \frac{\text{Vol}(E_0)}{\text{Vol}(P)}\right)$$

The per-iteration time is dominated by the time to find a bounding ellipsoid, which is polynomial.

4.2 Separation Oracles

Something to notice is that the ellipsoid algorithm does not require the linear program to have a polynomial number of constraints (it need not look at the linear program at all). It only requires a polynomial time *separation oracle*.

Definition 4. *A **separation oracle** reports whether a point is feasible, or else gives a violated constraint. Formally, given candidate solution x for $P = \{Ax \geq b\}$, show:*

1. *$x \in P$ or*
2. *$x \notin P$ and constraint a_i where:*

$$a_i x < b_i$$

If the number of constraints *is* polynomial, we could just check each one manually.

Example 1. Recall the path-based LP for maximum flow:

$$\begin{aligned} \max \quad & \sum_{p \in P(s,t)} f(p) \\ \text{s.t.} \quad & \sum_{p:(v,w)\in p} f(p) \leq u(v,w) \quad \forall (v,w) \in E \\ & f(p) \geq 0 \end{aligned} \qquad \begin{aligned} \min \quad & \sum_{(v,w)\in E} u(v,w)\ell(v,w) \\ \text{s.t.} \quad & \sum_{(v,w)\in p} \ell(v,w) \geq 1 \qquad \forall p \in P(s,t) \\ & \ell(v,w) \geq 0 \end{aligned}$$

The primal problem has an exponential number of variables, but a polynomial number of constraints (vice versa for the dual). An exponential number of dual constraints seems like a problem to verify, but we have a separation oracle.

A polynomial time separation oracle for the dual is the *shortest path* under $\ell(\cdot)$. If the length of the shortest path is < 1, then the shortest path is a violated dual constraint. Otherwise, when the length of the shortest path is ≥ 1, we know that *all* paths are at least length 1 under $\ell(cdot)$, and therefore the length function is feasible. Such a shortest path can be computed in polynomial time, using Djikstras or Bellman-Ford.

5 Interior Point Algorithms

Interior point algorithms stay inside the feasible space, and gradually approach the polytope boundary. These algorithms use a potential function to measure the duality gap and distance from the boundary, and a barrier function which makes the boundary unattractive. When the algorithm approaches the boundary, it maps the polyhedron to a new space where the boundary is farther away. Contrast to the simplex algorithm, which traces along the surface of the polytope. These are more practical than ellipsoid.

6 Summary

In this lecture, we gave a short overview of some of the current algorithms for linear programming. We also introduced the idea of separation oracles, which may exist despite exponential-sized linear programs.

References

[Kha80] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.