CS602 Applied Algorithms **2019-20 Sem II**

Lecture 18 - I: March 24

Scribe: Bismillah Hossainy *Lecturer: Rohit Gurjar*

1 Methods for Solving LP Problems

Till now, we have seen some primal-dual based methods to solve combinatorial optimization problems. Some of these problems were polynomial-time solvable like bipartite matching and some were NP-hard problems which we solved approximately, e.g., Steiner tree. Until now, we have not used any generic LP solvers, instead, we just used some ideas from LP duality and problem specific techniques. There are other paradigms to find either an exact or an approximate solution for combinatorial problems that actually use LP solvers (to be discussed in coming lectures). There are various methods to solve LP programs. In this lecture, we briefly discuss three of these methods in chronological order.

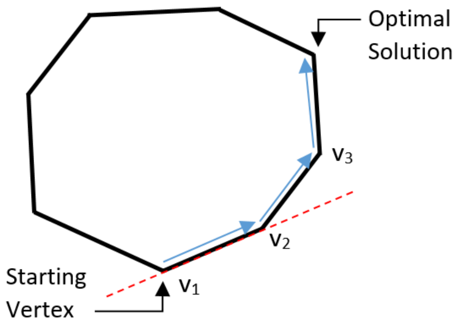
- Simplex Algorithm (1950s)
- Ellipsoid Method (1970s)
- Interior Point Method (1980s)

2 Simplex Algorithm

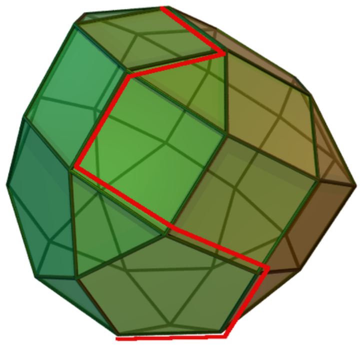
The simplex algorithm, the most famous and widely used method to solve linear programs, was developed in the 1950s by Dantzig. The Simplex method is robust; it solves any linear program, also detects if there is no feasible solution or unbounded optimal. Although the method is known to be really good at practice, it is not known to be polynomial time.
The Simplex algorithm works as follows:

- The method starts with a feasible vertex.
- Then, it moves from this vertex to a neighboring vertex (connected by an edge), at each step improving the value of the objective function.
- The algorithm terminates after a finite number of such transitions.

Figure-1 depicts the steps of the simplex algorithm. As seen on Figure-1.a, the process starts with the initial vertex of v_1, and then, it moves to vertices v_2, v_3, and so on until the algorithm finds the optimal solution.



a) Simplex Alg. in 2D



b) Simplex Alg. in 3D

Figure 1: The Simplex Algorithm moves vertex by vertex while improving the objective value.[2]

Let $Ax \leq b$ or $a_i^\top x \leq b$ for $1 \leq i \leq m$ be the given LP, where $x = (x_1, x_2, ...x_n)$. We know that it describes a polyhedron and any vertex of the polyhedron can be described by n tight constraints. There can be more than n tight constraints for a vertex[1], but for now, let us assume that every vertex has exactly n tight constraints. Let us first see how do we go to a neighboring vertex (two vertices are neighbors if they are connected by an edge). Suppose the starting vertex described by:

$$v_1 \quad \equiv \qquad a_i^\top x = b_i \text{ for } 1 \leq i \leq n$$

Now we want to move along with an edge. To do this we keep $n-1$ constraints tight and relax one of the constraints. Note that the solution of $n-1$ tight constraints is a line. Without loss of generality, suppose we relax the n^{th} constraint:

$$a_n^\top x = b_n \qquad \longleftarrow \textit{relax}$$

From the starting vertex, we should move in the direction which keeps the feasibility condition. That is, when we relax $a_n x = b_n$, we should maintain $a_n x \leq b_n$. We want to walk along the line given by

$$a_i^\top x = b_i, \quad 1 \leq i \leq n-1. \tag{1}$$

till the last point that is feasible. Take a generic point on this line satisfying $a_n^\top x = b_n - \lambda$ for $\lambda \geq 0$. Such a point x can be obtained by solving these n equations (note that they give a unique solution), which will be a function of λ. Now, take the maximum possible λ such that

$$a_i^\top x \leq b_i \text{ for each } n+1 \leq i \leq m.$$

Taking the maximum possible λ, one of these constraints ($(n+1)$-th to m-th) will become tight. Now, v_2 can be defined by this new set of n tight constraints.

For any vertex, there are n choices for the constraint to be relaxed, that means there are n possible edges incident a vertex. One can choose and walk along any one edge that increases the optimal value.

Claim: If a vertex v is not optimal, then there exists an edge on v that walking along it improves the objective value.

The above claim implies that the simplex algorithm takes finitely many steps to find the optimal vertex. Prove this claim as an exercise.

Tip: The argument is simply that from the starting vertex, we choose the next vertex, which strictly improves the objective value. How do we know if a vertex is optimal? A vertex is optimal when all edges on that vertex give a worse optimal value. As there are finitely many vertices to be chosen, so the simplex algorithm finishes in finitely many steps.

Running Time: While using the simplex algorithm to solve an LP problem, how many steps are required? Can we bound the total number of steps? One trivial number of bounds is the number of vertices, which can be exponential in n. Another bound we can get is from the objective function values. If we know that the objective function takes values between some lower limit ℓ and an upper limit L over the polytope. And at every step, the increment in the objective value is at least δ. Then we can get a bound of $(L-\ell)/\delta$ on the number of steps. In a more rigorous analysis, the number of steps would depend on the edge selection mechanism. It is possible that there is an edge selection rule that guarantees only polynomially many steps for any linear program. However no such rule is known till now.

Open: finding a rule to guarantee a polynomial bound on the number of steps in simplex for every possible LP.

There are two questions that we have left unanswered till now. One is the assumption we made in the beginning that every vertex has exactly n tight constraints. And the other is how to find the starting point for the simplex algorithm, that is, how to get a vertex of the polyhedron. Let us discuss them in order.

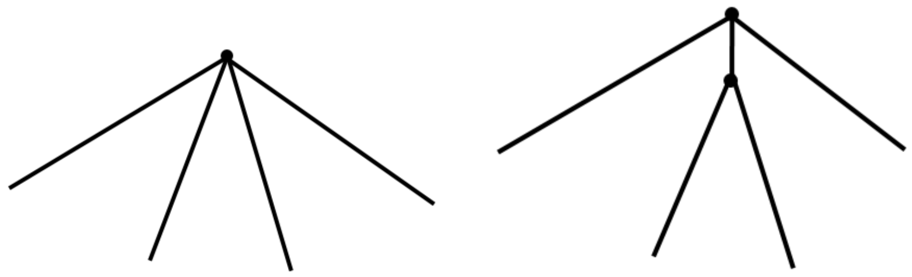
[1]for example, consider a pyramid in 3 dimensions; its apex has four planes meeting at it

Degenerate Case of Simplex Algorithm: There is no upper bound on the number of tight constraints a vertex of an n-dimensional polytope can have. That means there is no upper bound on the number of edges a vertex can have. In general, the number of edges can be more than n. Suppose that there are k tight constraints on a vertex, then there are $\binom{k}{n-1}$ choices of the constraints to be kept tight. Many of these choices can give an edge and thus, there can be exponentially many edges on a vertex. From these edges, we have to choose one edge which gives the best improvement value. How to go over all these possible edges, and which one to choose? There exist several ways of resolving this problem; one is *random perturbation.*

Random perturbation. Take the original set of constraints and add some small random number to b_i for every $1 \leq i \leq m$. So for the i^{th} constraint we will have

$$a_i^\top x \leq b_i + \epsilon_i$$

What this does is that it puts the bounding hyperplanes in a general position. When a set of hyperplanes in n-dimensional space are in a general position, it is not possible that there more than n hyperplanes that intersect on a point. This means that every vertex has n tight constraints and thus, only n edges will be incident on a vertex. For example, in a $3D$ polytope, if the bounding hyperplanes are in a general position, it is not possible to have four planes intersecting on a point. This process is illustrated in Figure-2. In Figure-2.a, four planes in 3-D are intersecting on a vertex. When we add some random ϵ_i to the i-th plane, this single vertex will be split into two vertices, giving us Figure-2.b. As seen in Figure-2.b, each vertex has only three planes that intersect on that vertex.



a) Before adding ϵ b) After adding ϵ on every plane

Figure 2: Random Perturbation

Because these added numbers are so small, the optimal value does not change very much. **Prove it as an exercise.** Therefore, random perturbation ensures that each vertex will have exactly n tight constraints, which imply that each vertex has exactly n edges. Now each of these edges can be checked by running the simplex algorithm to find the best promising edge.

Finding a Feasible Solution: Now we move on to the question of finding a starting vertex. Remember that finding a feasible solution is as hard as optimizing over an LP, so it is not immediately clear how to find one. The idea is to use the simplex algorithm itself to find a initial feasible solution. Consider a given linear program, say LP(1), in the form

$$\begin{aligned} Ax &= b \\ x &\geq 0. \end{aligned}$$

We can assume that $b \geq 0$ (if $b_i < 0$, we can multiply -1 in that constraint). Let us convert it to a new system by adding extra variables $z = \{z_i : 1 \leq i \leq m\}$, one for each constraint as follows.

$$\begin{aligned} Ax + z &= b \\ x &\geq 0 \\ z &\geq 0 \end{aligned}$$

First of all, it is easy to find an initial feasible solution for this system: take $x = 0$ and $z = b$. Now the feasible solution for the original system can be found using the following linear program LP(2)

$$\begin{aligned} \min \quad & \sum_{i=1}^{m} z_i \quad s.t. \\ & Ax + z = b \\ & x \geq 0 \\ & z \geq 0 \end{aligned}$$

Claim: If x is feasible for LP(1), then $(x, 0)$ is a minimizing point for LP(2). If there is no feasible point for LP(1), then the minimum value for the LP(2) is strictly more than zero.
Prove this claim as an exercise. To summarize:

- Finding the initial feasible solution for LP(2) is easy: $z = b, x = 0$.
- Use the simplex algorithm to find the optimal value for LP(2).
- Any optimal solution for LP(2) will give a feasible solution to LP(1).

References

[1] http://web.mit.edu/15.053/www/AMP-Chapter-02.pdf

[2] https://en.wikipedia.org/wiki/Simplex_algorithm#Overview