CS 270: Combinatorial Algorithms and Data Structures Spring 2023

Lecture 21 — April 4, 2023

Prof. Jelani Nelson *Scribe: Hanzhe Wu, Chethan Bhateja*

1 Overview

In today's lecture we will start to talk about linear programming. More specifically, we plan to discuss

- Simplex method
- Strong duality
- Complementary slackness

Remark. We will show in the next lecture how to prove strong duality through the simplex method (which would be a natural corollary). Strong duality can also be proved via Farkas' lemma.

2 Linear Programming

Recall that linear programs optimize a linear function subject to linear constraints. In general, LPs can be written with inequality constraints in canonical form

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

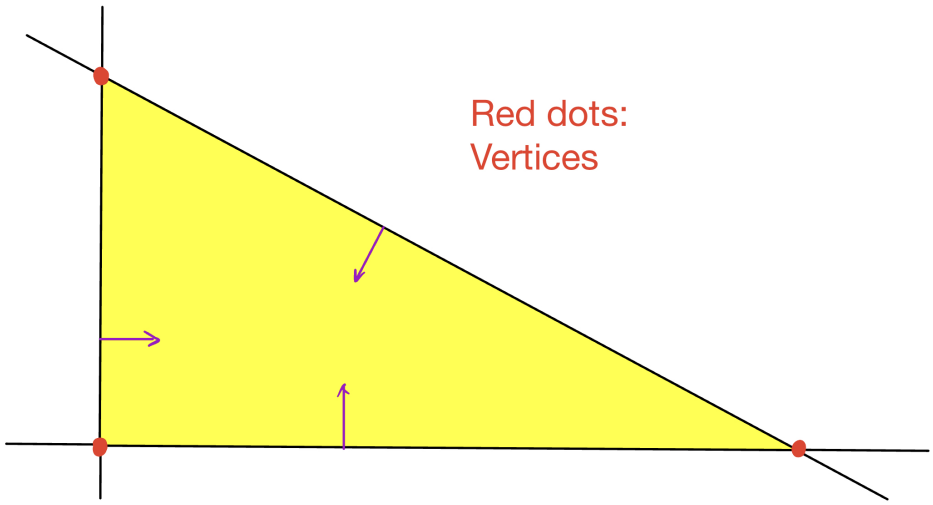
We can also write an LP in standard form with equality constraints, which is how it is typically inputted into the simplex method.

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

where $A \in \mathbb{R}^{m \times n}$, $n \geq m$.
In fact, any LP can be rewritten in standard form:

- For constraints $\langle a_i, x \rangle \leq b_i$, we can introduce slack variables s_i and rewrite them as $\langle a_i, x \rangle + s_i = b_i$, $s_i \geq 0$.
- For free variables x_i, we can define $x_i^+, x_i^- \geq 0$, and replace x_i with $x_i^+ - x_i^-$.
- If $n < m$, add dummy variables to make $n \geq m$.

At some points, we will also assume the rows of A are linearly independent. This is reasonable, since redundant and contradictory constraints can be found efficiently via row reduction.
The figure below illustrates a simple LP in canonical form.



3 Simplex Method

3.1 General Description

Key to the simplex method is that the optimum OPT is always achieved at a vertex, which we will define shortly. The algorithm works roughly as follows:

1. Find starting vertex $\vec{x}_0$.
2. While $\vec{x}_i$ is sub-optimal, greedily move to better neighboring vertex $\vec{x}_{i+1}$.
3. HALT, return $\vec{x}_T$.

Remark 3.1. Actually, the first step to find a vertex is as hard as solving the LP! There is an efficient reduction from optimizing an LP $\longrightarrow$ finding a feasible x for the LP via binary search on OPT. At each iteration, we can guess that $\mathsf{OPT} \leq \alpha$, add the constraint $c^\intercal x \leq \alpha$ to form the new LP below, and find a feasible x for this LP.

$$\begin{array}{lrcl} \min & \mathbf{0}^\top x & & \\ \text{s.t.} & Ax & = & b \\ & c^\top x & \leq & \alpha \\ & x & \geq & 0 \end{array}$$

For the time complexity, notice that if the result has ℓ-bit precision, then we will do $O(\ell)$ rounds of binary search. In fact, our inputs A, b, c have only finite precision. If all of them have $\leq \ell$ bits precision, then the optimal solution would only have $\text{poly}(nm\ell)$ bits precision. Thus, once α has enough precision (which would not take long), we can claim that we have found the optimal value α.

To further analyze LPs and the simplex algorithm, we need the following definitions.

Definition 3.2 (Feasible Set)**.** The feasible set P is the set of all x satisfying all constraints. i.e., $P = \{x : Ax = b, x \geq 0\}$.

Definition 3.3 (Feasible)**.** A point x is feasible if $x \in P$.

Definition 3.4 (LP Feasibility)**.** An LP is feasible if $P \neq \emptyset$.

Definition 3.5 (Bounded)**.** An LP is bounded if $\mathsf{OPT} > -\infty$.

Definition 3.6 (Vertex)**.** $x \in P$ is a vertex if $\begin{cases} x + y \in P \\ x - y \in P \end{cases} \implies y = 0$

Remark. The definition of vertex meets our intuition – for a vertex, we cannot move in opposite directions while staying feasible!

3.2 Finding Starting Vertex

Next, we will try to find a starting vertex. We will do this by formulating another LP as follows:

$$\begin{aligned} \min \quad & t \\ \text{s.t.} \quad & Ax &&= (1-t)b \\ & x, t &&\geq 0 \\ & t &&\leq 1 \end{aligned}$$

This LP is not in the standard form. It is easy to transform the first group of constraints to standard form, while for the last constraint, we can add a slack variable s_t and rewrite it as $\begin{cases} t + s_t &= 1 \\ s_t &\geq 0 \end{cases}$.

Note that the optimum is $t = 0 \iff$ the original LP is feasible.

Now we want a starting vertex to run the simplex algorithm for the new LP. Consider $x = \vec{0}$, $t = 1$, and $s_t = 0$. We can check that these values are feasible for the LP. Next, we show the $(n+2)$-dimensional vector of $(\underbrace{\vec{x}}_{n \text{ vars}}, t, s_t) = (\vec{0}, 1, 0)$ is a vertex. Intuitively, this is because we are at the edge of the feasible set for all coordinates.

Let $y = (y_1, \cdots, y_n, y_{n+1}, y_{n+2})$ be the vector we add and subtract as in the vertex definition.

- y cannot have support (nonzero values) in the first n coordinates. Otherwise, if $y_i \neq 0$, either $x_i + y_i$ or $x_i - y_i$ would be < 0, making the ith coordinate < 0 and violating feasibility.
- y cannot have support in the $(n+1)$th coordinate. Otherwise, either $t + y_{n+1}$ or $t - y_{n+1}$ would be > 1, making the $(n+1)$th coordinate > 1 and violating feasibility.
- y cannot have support in the $(n+2)$th coordinate. Otherwise, either $s_t + y_{n+2}$ or $s_t - y_{n+2}$ would be < 0, making the $(n+2)$th coordinate < 0 and violating feasibility.

Thus, $y = 0$, and $(\vec{x}, t, s_t) = (\vec{0}, 1, 0)$ is a vertex.

3.3 Theorems for Simplex Algorithm

Now we turn to the justification of the simplex algorithm. Why does it work? The following theorems will convince us step by step.

Firstly, an important feature of the simplex algorithm is that it always "jumps" among vertices seeking optimal values. Thus, it is important for the LP to have an optimal value lying in the vertices, as stated in the next claim. Notice that we are now dealing with a minimization problem.

Claim 3.7. If an LP is bounded and feasible, then $\forall x \in P$, $\exists$ a vertex $x' \in P$, s.t. $c^\top x' \leq c^\top x$.

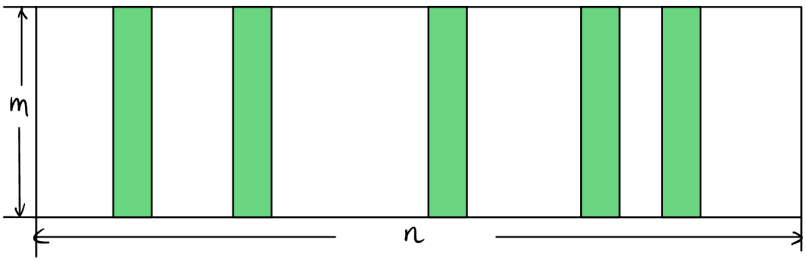
Proof. Assume, for contradiction, that x' is not a vertex, then $\exists\, y \neq 0$, s.t. $x+y, x-y \in P$, i.e., $\underbrace{A(x+y)=b, A(x-y)=b}_{Ay=0}$, and $x+y, x-y \geq 0$. Without loss of generality, let $c^\top y \leq 0$ (we may rename $y \leftarrow -y$). If $c^\top y = 0$, as $\exists\, j$, s.t. $y_j \neq 0$, then WLOG, $\exists\, j$, s.t. $y_j < 0$ (Similarly, we can rename y if otherwise).
Now consider two cases of y:
Case 1: $\exists\, j$, s.t. $y_j < 0$. Note that as $x+y, x-y \geq 0$, $\text{supp}(y) \subseteq \text{supp}(x)$, i.e., $x_i = 0 \Rightarrow y_i = 0$. Consider $x + ty, t \geq 0$. If t is small enough, then adding ty to x would not violate any constraints, since if $y_j \neq 0$, then $x_j > 0$. Thus, we can gradually increase t from 0 and stop when some $x_i = 0$. We pick $t^* = \min_{i:y_i<0} |\frac{x_i}{y_i}|$, and change x to $x + t^* y$.
Case 2: $\forall j$, $y_j \geq 0$ (i.e. $y \geq 0$). In this case, we can assume $c^\top y < 0$ since $y \geq 0$ and $c^\intercal y \leq 0$. Note that $x + ty \in P$, $\forall t \geq 0$, then $\mathsf{OPT} = -\infty$. Thus, case 2 contradicts our premise that the LP is bounded and is impossible.
Notice that whenever we are in case 1, one more coordinate will be set to 0. We can repeat case 1 over and over again, setting more and more coordinates x_i to 0 and making x more vertex-like. When enough x_i are 0 and at the boundary of the feasible region, there will no longer exist a nonzero y to perturb x, making x a vertex. □

Next, we will define the concept of basis, and show an equivalent condition of a point being a vertex and the columns of A corresponding to the basis of that point in a claim.

Definition 3.8 (Basis)**.** Given a vertex $x \in P$, the basis of x is $B_x = \{j \in [n] : x_j > 0\} = \text{supp}(x)$.

Claim 3.9. $\boxed{x \in P \text{ is a vertex}} \iff \boxed{\text{columns } A_{B_x} \text{ are linearly independent}}$, where A_S denotes A restricted to the columns S.

By the claim above, since $m \leq n$, a vertex can have at most m columns. Hence, a way to find a vertex is to take m independent columns as the basis, as illustrated in the figure below.



Then, as $Ax = b$,

$$Ax = \sum_{i=1}^{n} x_i A_i = \sum_{i \in B_x} x_i A_i \implies Ax = A_{B_x} x_B = b \implies x_B = A_{B_x}^{-1} b$$

where x_B is the vector x restricted to the indices in B_x. Thus, to find x_B, we can just take the inverse of the restricted columns of the matrix after taking the basis. The other parts of x are just 0's. Next, we will prove the claim.

Proof. We will show both directions by contraposition. First, we will show $\boxed{x \in P \text{ is not a vertex}}$ $\implies$ $\boxed{\text{columns } (A_{B_x}) \text{ are linearly dependent}}$.

If $x \in P$ is not a vertex, then $\exists\, y \neq 0$, s.t. $\begin{matrix} A(x+y) = b \\ A(x-y) = b \end{matrix}$ and $\begin{matrix} x+y \geq 0 \\ x-y \geq 0 \end{matrix}$

Thus, $Ay = A_{B_y} y' = 0$, and $B_y \subseteq B_x$. Hence, $A_{B_x} y'' = 0$, and thus, columns of A_{B_x} are linearly dependent.

Remark. y is an n-dimensional vector, y' is a $|B_y|$-dimensional vector with all the zero elements "chopped off" from y. y'' is a $|B_x|$-dimensional vector generated from B_y and adding back several 0 entries. Thus, since $y \neq 0$, by definition, $y' \neq 0$, and thus $y'' \neq 0$ and columns of A_{B_x} are linearly dependent.

Next, we will show $\boxed{\text{columns } (A_{B_x}) \text{ are linearly dependent}}$ $\implies$ $\boxed{x \in P \text{ is not a vertex}}$.

Columns (A_{B_x}) are linearly dependent $\implies$ $\exists y \neq 0$, s.t. $A_{B_x} y = 0$. Thus, $\exists\, y' \in \mathbb{R}^n$, s.t. $Ay' = 0$, and $\text{supp}(y') \subseteq B_x$(we can achieve this by padding 0's to other entries). Thus, $y_i \neq 0 \Rightarrow x_i > 0$.

Hence, $\exists\, t > 0$, s.t. $\begin{cases} x + ty' \in P \\ x - ty' \in P \end{cases}$ $\implies$ x is not a vertex. □

3.4 The Simplex algorithm

Now, we will talk about the algorithm itself. The first thing we need to note is that for any vertex x with $|B_x| < m$, we need to artificially add more linearly independent columns from A to make $|B_x| = m$. This may cause problems, as we will see in the last part of today's lecture.
Next, we can (finally) give a more detailed version of the simplex algorithm.

1. Start at some basis B.
2. while $\exists$ a better neighbor, move there
3. HALT.

Naturally, the next questions we may ask are "what does 'a better neighbor' mean?" and "when should we halt?". To answer them, if we are given a particular basis B, we may rewrite the LP as

$$\begin{matrix} \min & c_B^\top x_B + c_N^\top x_N \\ \text{s.t.} & \boxed{A_B x_B + A_N x_N = b} \\ & x_B, x_N \geq 0 \end{matrix}$$

where $N := [n] \backslash B$. Hence,

$$x_B = A_B^{-1} b - A_B^{-1} A_N x_N \implies \mathsf{cost} = \underbrace{c_B^\top A_B^{-1} b}_{\text{const}} - c_B^\top A_B^{-1} A_N x_N + c_N^\top x_N$$

Therefore, we need to optimize (minimize) $\underbrace{(c_N - A_N^\top \left(A_B^{-1}\right)^\top c_B)^\top}_{\tilde{c}_N} x_N$, where x_N is currently all 0.

Returning to our questions, now we know that "$\exists$ a better neighbor" means $\exists\, j$, s.t. $(\tilde{c}_N)_j < 0$, and we halt if every entry of $\tilde{c}_N \geq 0$.

If we look carefully at the algorithm, we can find that changing entries in x_N would also change the entries in x_B. In fact, each time we throw an index j into the basis, some indices in B would become 0, and then we will kick them out from the basis. If there are multiple j's that can be chosen to throw into B, we may choose one of them freely or by some rules.
However, as we artificially add some entries to B when $|B_x| < m$, these entries may already be 0. Thus, for some $j \in N$, x_j may not be jacked up (increased) at all! In this case, we will add j into B and kick out some "bad" entry from B.
That is still not the full story – we may get into an ∞-loop if we don't do this wisely! The good news is that there are "pivot rules" for us to choose the index to throw into the basis when multiple j's can be chosen, and the entry to kick out from the basis when some of them are 0. For instance, we can use the Bland's rule [Bla77] discovered in the 1970s. Such rules would guarantee the algorithm would terminate, but the bad news is that all known pivot rules would take exponential time in the worse case!

4 Conclusion

The simplex method was first discovered by George Dantzig in 1947 [Dan51]. There is a famous story on Dantzig's solving open problems related to simplex algorithm because he mistakenly regarded them as homework when he was a Ph.D. student in UC Berkeley.
In the next lecture, we will cover strong duality and its proof. It states that $\exists$ dual feasible y s.t. $c^\top x = b^\top y$. The proof will be based on writing down y with basis B.

References

[Bla77] Robert G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2(2):103–107, 1977.

[Dan51] George B. Dantzig Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.