Advanced Algorithms

Lectures by Ely Porat
Summary by Ari Feiglin (ari.feiglin@gmail.com)

Contents

1 Linear Programming

1.1 Linear Programs

Suppose we have a pizzeria which sells different types of pizzas, each with an assortment of toppings. This pizzeria has one-thousand units of dough, 500 units of sauce, and 800 units of cheese. The pizzeria sells the following types of pizzas, which each require the following units of toppings and have the following prices:

	Regular Pizza	**Thin Pizza**	**Extra-Cheese Pizza**
Dough	2	1	2
Sauce	2	3	2
Cheese	1	2	3
	\$25	\$30	\$40

So if we sell x_1 regular pizzas, x_2 thin pizzas, and x_3 extra-cheese pizzas, we must have the following constraints:

$$2x_1 + x_2 + 2x_3 \leq 1000, \qquad 2x_1 + 3x_2 + 2x_3 \leq 500, \qquad x_1 + 2x_2 + 3x_3 \leq 800$$

And we want to maximize our total sales, i.e. maximize $25x_1 + 30x_2 + 40x_3$ where $x_1, x_2, x_3 \geq 0$.
This is an example of *linear programming* (LP).

1.1.1 Definition

A **Linear Program** is the problem of maximizing some objective expression of the form

$$f(x_1, \ldots, x_n \,;\, c_1, \ldots, c_n) = \sum_{i=1}^{n} c_i x_i$$

for some parameters c_i under the constraints

$$\sum_{j=1}^{n} \alpha_{i,j} x_i \leq \beta_i$$

for $1 \leq i \leq m$ where m is the number of constraints.

Notice that if we want to have the constraint $\sum_{j=1}^{n} \alpha_{i,j} x_i = \beta_i$, this is just the same as having two constraints $\sum_{j=1}^{n} \alpha_{i,j} x_i \leq \beta_i$ and $\sum_{j=1}^{n} \alpha_{i,j} x_i \geq \beta_i$. And if we want the constraint $\sum_{j=1}^{n} \alpha_{i,j} x_i \geq \beta_i$, this is equivalent to $\sum_{j=1}^{n} (-\alpha_{i,j} x_i) \leq -\beta_i$.
Now say we want to *minimize* $f(x_1, \ldots, x_n \,;\, c_1, \ldots, c_n)$, this is just the same as maximizing the expression $f(x_1, \ldots, x_n \,;\, -c_1, \ldots, -c_n)$.

1.1.2 Definition

An LP is in **standard form** if it has the constraints $x_i \geq 0$ for every variable x_i.

This seems to weaken the strength of LP, but in fact every general LP can be converted to an equivalent LP in standard form. Suppose we have the LP $f(x_1, \ldots, x_n \,;\, c_1, \ldots, c_n)$ with constraints using $\alpha_{11}, \ldots, \alpha_{n,m}$ and $\beta_1, \ldots, \beta_m$. Then we can define

$$f'(x_1^+, x_1^-, \ldots, x_n^+, x_n^-) = \sum_{i=1}^{n} c_i (x_i^+ - x_i^-)$$

and the constraints

$$\sum_{j=1}^{n} \alpha_{i,j} (x_i^+ - x_i^-) \leq \beta_i \qquad (1 \leq i \leq m)$$

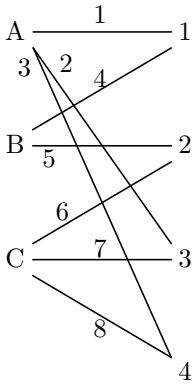
along with $x_i^+, x_i^- \geq 0$. This is equivalent, as we can then just define $x_i = x_i^+ - x_i^-$, we have split x_i into is positive and negative parts.

Thus given an LP in standard form, we can write it as a problem of the form

$$\begin{aligned} \text{Maximize} \quad & \vec{c}^{\top} \cdot \vec{x} \\ \text{Such that} \quad & A\vec{x} \leq \vec{b} \\ & \vec{x} \geq \vec{0} \end{aligned}$$

where $\vec{c}$ is the vector of coefficients $(c_1, \ldots, c_n)^{\top}$, $\vec{x}$ is the vector of variables $(x_1, \ldots, x_n)^{\top}$, and A is the matrix such that $A_{ij} = \alpha_{ij}$. We write $\vec{a} \leq \vec{b}$ to mean $a_i \leq b_i$ for every i.

Example: Suppose we have the following graph (the edges are labelled), and we'd like to find the maximum number of matches we can find.



The LP form of this would be maximizing $\sum_{i=1}^{8} x_i$ (where x_i denotes whether or not the ith edge was chosen), with the constraints ($x_i \geq 0$ is implicit)

$$\begin{aligned} x_1 + x_2 + x_3 &\leq 1 \\ x_4 + x_5 &\leq 1 \\ x_6 + x_7 + x_8 &\leq 1 \\ x_1 + x_4 &\leq 1 \\ x_5 + x_6 &\leq 1 \\ x_2 + x_7 &\leq 1 \\ x_3 + x_8 &\leq 1 \end{aligned}$$

But we also want the constraint that x_i be an integer. This is called *Integer Linear Programming*. This is an NP-hard problem, as we will see in the following example. ◊

Example: Now let us look at another example: the problem of *Vertex Covers* (VC). Given a graph $G = (V, E)$, we want to find the smallest vertex cover, which is a set of vertices S such that every $v \in V$ has a neighbor in S. Now, we can use LP as follows: associate each vertex $v \in V$ with a variable x_v. Then we want to minimize $\sum_{v \in V} x_v$. And we add the constraints that for every $(u, v) \in E$, $x_u + x_v \geq 1$ (i.e. either u or v is in S). We can solve this using integer linear programming, and since this is an NP-complete problem (whether G has a VC of a particular size), so too is integer linear programming.

Using LP won't necessarily give us a valid solution to the problem since we can have that x_v be a non-integer. We only know that the answer LP gives us is at most VC_{opt} (the optimal solution), since the optimal solution satisfies the constraints. In fact, if we define $\text{Sol} = \left\{ v \;\middle|\; x_v \geq \frac{1}{2} \right\}$ then we have that

$$\text{LP} \leq \text{VC}_{opt} \leq |\text{Sol}| \leq 2\text{LP}$$

This is because Sol must be a vertex cover: if there exists a $v \in V$ with no neighbors in Sol, this means that $x_v < \frac{1}{2}$ and $x_u < \frac{1}{2}$ for every $(v, u) \in E$. But then we can just set $x_v = \frac{1}{2}$ and this will satisfy the constraints. And if we multiply LP by 2, then every vector in Sol gets multiplied by 2, and so summing them gives Sol. ◊

1.1.3 Definition

The **slack form** of an LP is one of the form:

$$\begin{aligned} \text{Maximize} \quad & \vec{c}^{\top} \cdot \vec{x} \\ \text{Such that} \quad & A\vec{x} = \vec{b} \\ & \vec{x} \geq \vec{0} \end{aligned}$$

Every LP in general form can be brought to one in slack form by adding variables. For every constraint $\sum_{j=1}^{n} \alpha_{i,j} x_j \leq \beta_i$ we add a basic variable s_i and alter the constraint to be $\sum_{j=1}^{n} \alpha_{i,j} x_j + s_i = \beta_i$. Since we must also add the constraint $s_i \geq 0$, this is equivalent. The coefficients of the basic variables in the expression we optimize is of course zero.

We will also assume that in slack form, $\vec{b} \geq \vec{0}$ as otherwise there may be no solution.

1.2 The Simplex Algorithm

First we begin by defining a few mathematical concepts.

1.2.1 Definition

A **convex set** $S \subseteq \mathbb{R}^n$ is a set such that for every $x, y \in S$, $\lambda x + (1-\lambda)y \in S$ for all $\lambda \in [0,1]$. The **convex hull** of a set of points $\{p_1, \ldots, p_m\}$ is the smallest convex set containing these points. It can be shown that the convex hull is

$$[p_1, \ldots, p_n] = \left\{ \sum_{i=1}^{m} \lambda_i p_i \;\middle|\; \sum_{i=1}^{m} \lambda_i = 1 \right\}$$

An **extreme point** of a convex set is a point $x \in S$ such that there exists no $a, b \in S$ such that x lies on the open segment between a and b. I.e. there exist no $a, b \in S$, $\lambda \in (0,1)$ such that $x = \lambda a + (1-\lambda)b$. Two extreme points x, y are **adjacent** if the line connecting them is not intersected by any other line between two points not on the line between x and y.

Notice that if $[p_1, \ldots, p_n]$ is a convex hull, then every extreme point is one of the p_is.

1.2.2 Definition

A **convex polytope** is the region of some n-dimensional space defined by $A\vec{x} \leq \vec{b}$. The extreme points of a convex polytope are called **vertices** and there are finitely many of them (in fact a convex polytope is the convex hull of its vertices).

Notice that the space of *feasible solutions* to an LP problem under the constraint $A\vec{x} \leq \vec{b}$ and $\vec{x} \geq 0$ forms a convex polytope (it is obviously a polytope, convexity is easy to verify). We will use the following results for constructing an algorithm to find the maximum of some objective $\vec{c}^{\top}\vec{x}$.

1.2.3 Lemma

Let P be a convex polytope with vertices $v_1, \ldots, v_n$ and f be a convex function over P. Then f attains its maximum at one of the vertices.

Proof: suppose x is the maximum of f, then since it is in P, it can be written as $x = \sum_{i=1}^{n} \lambda_i v_i$ for $\sum_{i=1}^{n} \lambda_i = 1$ and $\lambda_i \geq 0$. So by convexity

$$f(x) = f\left(\sum_{i=1}^{n} \lambda_i v_i\right) \leq \sum_{i=1}^{n} \lambda_i f(v_i) \leq \sum_{i=1}^{n} \lambda_i f(x) = f(x)$$

Thus $f(x) = \sum_{i=1}^{n} \lambda_i f(v_i)$, but $f(v_i) \leq f(x)$ for all i so $f(v_i) = f(x)$ for all i with $\lambda_i \neq 0$, meaning v_i must be a maximum. ■

So to find the maximum of the objective, it is sufficient to find its maximum over the vertices of the polyope defined by $A\vec{x} \leq \vec{b}$. But there are many vertices, so how can we effectively find the maximum?

1.2.4 Lemma

Let P be a convex polytope and a linear function $f(\vec{x}) = \vec{c}^\top \vec{x}$, then if a vertex is larger than all its adjacent vertices, it is a maximum.

We will prove this later. But this tells us that if we start from a vertex and traverse to adjacent vertices such that each one is larger than the next, we will eventually reach a vertex greater than all its neighbors and this must be a maximum.

Notice that when we consider $A\vec{x} = \vec{b}$, we can assume that A has full row rank, since otherwise there is no solution or A has extra equations which we can remove.

1.2.5 Definition

A **basis** of the polytope defined by $A\vec{x} \leq \vec{b}$ is a set of indexes B such that when we take the matrix whose columns are the columns of A whose index are in B, denoted A_B, A_B is invertible.

If we have the boundary of a convex polytope $A\vec{x} = \vec{b}$ where $A \in \mathbb{R}^{m \times n}$, then the vertices are the vectors $\vec{x}$ where $x_i = 0$ for $i \notin B$ where B is some basis. Call such vectors *basic feasible solutions* (bfs), we will show later why these are precisely the vertices of the polytope. For each basis B, such a point is unique, since they must have $A_B\vec{x} = \vec{b}$ and so $\vec{x} = A_B^{-1}\vec{x}$.

These are indeed vertices: we prove this by showing that they are extreme points. Suppose that $\vec{\alpha}, \vec{\beta}$ are two points in the polytope such that $\lambda\vec{\alpha} + (1-\lambda)\vec{\beta} = \vec{x}$ for $\lambda \in (0,1)$. Then on one hand

$$A(\lambda\vec{\alpha} + (1-\lambda)\vec{\beta}) = \lambda A\vec{\alpha} + (1-\lambda)A\vec{\beta} \leq \lambda\vec{b} + (1-\lambda)\vec{b} = \vec{b}$$

but on the other hand this must be an equality since $\lambda\vec{\alpha} + (1-\lambda)\vec{\beta} = \vec{x}$. So $A\vec{\alpha} = A\vec{\beta} = \vec{b}$. For every i for which $x_i = 0$ we must have $\alpha_i = \beta_i = 0$ since $\lambda\alpha_i + (1-\lambda)\beta_i = 0$ and $\alpha_i, \beta_i \geq 0$. But we said that such points are unique, so $\vec{\alpha} = \vec{\beta} = \vec{x}$, so $\vec{x}$ is an extreme point.

We also make the following observation: maximizing $c^\top\vec{x}$ by $A\vec{x} = \vec{b}$ is the same as finding a maximal v such that there exists a $\vec{x}$ where

$$\begin{pmatrix} A & 0 & b \\ -c^\top & 1 & 0 \end{pmatrix} \begin{pmatrix} \vec{x} \\ v \\ -1 \end{pmatrix} = 0$$

The left matrix is called the *tableau* of the LP.

Now suppose B is a basis for A, then if we multiply by $A_B^{-1} \oplus 1$ (which is legal since it is invertible) we get that we can solve for the tableau

$$\begin{pmatrix} I \mid_B A' & 0 & A_B^{-1}b \\ -c^\top & 1 & 0 \end{pmatrix}$$

Where $I \mid_B A'$ means that the columns of matrix I are inserted at indexes in B to the matrix A'. Let us assume that $I \mid_B A' = (I\ A')$, then we have a tableau of the form

$$\begin{pmatrix} I & A' & 0 & \bar{b} \\ -c_1^\top & -c_2^\top & 1 & 0 \end{pmatrix}$$

Performing some row-reduction gives us

$$\begin{pmatrix} I & A' & 0 & \bar{b} \\ 0 & -\bar{c}_2^\top & 1 & z_B \end{pmatrix}$$

Recall that the vertex associated with the basis B has zeroes when $i \notin B$, so multiplying by $(\vec{x}\ v\ -1)^\top$ gives us

$$\begin{pmatrix} I & A' & 0 & \bar{b} \\ 0 & -\bar{c}_2^\top & 1 & z_B \end{pmatrix} \begin{pmatrix} \vec{x} \\ v \\ -1 \end{pmatrix} = \begin{pmatrix} \vec{x}_B - \bar{b} \\ v - z_B \end{pmatrix}$$

So we have that $\vec{x}_B = \bar{b}$ gives us the value $v = z_B$ ($\vec{x}_B$ is the vector $\vec{x}$ with indexes not in B removed). What this tells us that if we evaluate $x_i = \bar{b}_i$ for $i \in B$ and $x_i = 0$ for $i \notin B$ we get a vertex of the polytope, and its value is z_B.

Using the second lemma, that a vertex is a maximum if and only if its value is greater than all its neighbors, we can continue this algorithm. How do we determine if two vertices are adjacent? We will answer this through the following lemmas.

1.2.6 Lemma

Let B_1, B_2 be two bases which differ in a single index, and x, y be the bfs-s corresponding to the bases. Then for every point a in the boundary of the polytope (meaning $Aa = b$), if $a_i = 0$ for every $i \notin B_1 \cup B_2$, a lies on the line between x and y.

Proof: we will show that the line passing through a and x contains y. That is, there exists a λ such that $\lambda a + (1-\lambda)x = y$. Suppose that $B_1 \setminus B_2 = \{j\}$ and $B_2 \setminus B_1 = \{k\}$. Then it is sufficient that $\lambda a_i + (1-\lambda)x_i = 0$ for $i \notin B_2$, since $A(\lambda a + (1-\lambda)x) = b$ and such a point is uniquely y. Therefore it is sufficient that $\lambda a_j + (1-\lambda)x_j = 0$, i.e. $\lambda(a_j - x_j) = -a_j$. This has a solution when $x_j \neq a_j$.

Thus the only issue arises when $x_j = a_j$. Let us denote $A_i = A_{B_i}$, then

$$Aa = b \implies A_1^{-1}Aa = x \implies (I \mid_B A_1^{-1}A_{B_1^c})a = x$$

Since $a_i = 0$ for $i \notin B_1 \cup B_2$, we have that

$$a_{B_1} + a_k A_1^{-1} C_k(A) = a_{B_1} + a_k C_k(A_1^{-1}A) = x$$

Thus $x_j = a_j + a_k[A_1^{-1}A]_{jk}$, so $a_k[A_1^{-1}A]_{jk} = 0$. If $a_k = 0$ then $a = x$ by uniqueness. Otherwise, $[A_1^{-1}A]_{jk} = 0$ why can't this happen? ■

Generalizing this lemma, we get the converse of what we stated earlier: a point is a vertex iff it is a bfs. This is since we can generalize this lemma to say that if $B_1, \ldots, B_k$ differ each by a point and $Aa = b$ such that $a_i = 0$ for $i \notin \bigcup_j B_j$, then a lies on the convex hull defined by $x_1, \ldots, x_k$ where x_i is the bfs defined by by B_i. So the polytope is the convex hull of all the bfs-s, and thus each extreme point must be a bfs.

1.2.7 Lemma

Two bfs-s are adjacent iff their bases differ by a single point.

Proof: suppose that B_1, B_2 are bases which differ by a single index. Let x, y be the corresponding bfs-s, and suppose that α, β are two points in the polytope such that the open segment between them intersects the line segment between x and y. Then there exists $\gamma \in (0,1), \lambda \in [0,1]$ such that

$$\lambda x + (1-\lambda)y = \gamma\alpha + (1-\gamma)\beta$$

this means that for $i \notin B_1 \cup B_2$, $\gamma\alpha_i + (1-\gamma)\beta_i = 0$, since $\alpha, \beta \geq 0$ this means $\alpha_i = \beta_i = 0$. But then by the above lemma, they are both on the line segment of x and y, so x and y are adjacent.

Show converse ■

Proof (of lemm 1.2.4):

So now let us discuss the significance of these lemmas on our algorithm. A current naive algorithm would be to iterate over all the bases B, and multiply the tableau by A_B^{-1}. Notice that this is just equivalent to row-reducing the tableau at the columns indicated by B. Recall that if we have a basis B, our tableau will be in the following form:

$$\begin{pmatrix} I \mid_B A' & 0 & \bar{b} \\ 0 \mid_B -\bar{c}^\top & 1 & z_B \end{pmatrix}$$

and the value at the vertex x_B (the bfs defined by B) is z_B. Choosing a basis B' which differs from B by a single index and then looking at the resulting tableau is the same as choosing a column from A' and using an element from that column as a pivot. The column we choose is what will determine the *entering index* (or entering variable), and the row we choose will determine which column from B will leave (the *leaving index* or variable).

Suppose we choose the index i to enter and the index j to leave, i.e. the column i and the row j. Then to row reduce the final row, we must add $\bar{c}_j / A'_{ji}$, resulting in a change of z_B to $z_{B'} = z_B + \frac{\bar{c}_j}{A'_{ji}}\bar{b}_j$. So we want to find the i, j which maximize $\frac{\bar{c}_j \cdot \bar{b}_j}{A'_{ji}}$. Notice that if all of c is negative, then this means that our current bfs is larger than all its neighbors and so it is maximal by an above lemma.

But we also need our new solution to be feasible, that is we cannot have that any b_k turns negative. The new value of b_k is, when i, j are chosen, $b_k - \frac{b_j}{A_{ji}}A_{ki}$. So we must have that

$$\frac{b_k}{A_{ki}} \geq \frac{b_j}{A_{ji}}$$

So after choosing j, we must choose i to minimize b_j/A_{ji}.

So what we do is we choose the column j such that $\bar{c}_j$ is maximal (the most negative value in the bottom row) (this is to find the optimal neighbor), and then choose the row i such that b_j/A_{ji} is minimal (to ensure feasibility).

This is the simplex algorithm. It is correct since at the worst case we visit every vertex, but that's still a finite number of vertices. Why is it called the *simplex* algorithm? Who knows, in my opinion a more fitting name would be "the convex polytope algorithm", but that's a bit of a mouthful.

1.3 The Simplex Algorithm; In Short

Suppose we want to solve the following LP in slack form: maximize $c^\top x$ constrained to $Ax = b$ where $A \in \mathbb{R}^{m\times n}$. Then we write out the *initial tableau*:

$$\begin{pmatrix} A & 0 & b \\ -c^\top & 1 & 0 \end{pmatrix}$$

If A contains the $m \times m$ identity matrix, then say that the tableau is in *canonical form*. Changing names of variables (i.e. column swapping), this means that the tableau is of the form

$$\begin{pmatrix} I & A & 0 & b \\ -d^\top & -c^\top & 1 & 0 \end{pmatrix}$$

Row reducing gives us

$$\begin{pmatrix} I & A & 0 & b \\ 0 & -c^\top & 1 & z_B \end{pmatrix}$$

z_B is a feasible solution to the problem when we set the variables in the columns corresponding to the identity matrix to the values of b and all the other variables to zero.

If the tableau is not in canonical form, we can find m independent columns and row reduce those to get the identity.

If c has no positive values, then we finish and z_B is the maximum value. Otherwise, choose the column i for which c_i is maximal. Then choose the row j for which $\frac{b_j}{A_{ji}}$ is minimal, and use the index (j, i) as a pivot in row reducing. This will give (after swapping columns)

$$\begin{pmatrix} I & \bar{A} & 0 & \bar{b} \\ 0 & -\bar{c}^\top & 1 & z_{B'} \end{pmatrix}$$

where $z_{B'} > z_B$, and this is another feasible solution. Continue iteratively choosing a pivot until c has no positive values.

Example: suppose we want to maximize

$$2x + 3y + 4z$$

subject to

$$\begin{aligned} 3x + 2y + z &\leq 10 \\ 2x + 5y + 3z &\leq 15 \\ x, y, z &\geq 0 \end{aligned}$$

Adding slack variables, this is equivalent to

$$\begin{aligned} 3x + 2y + z + s_1 &= 10 \\ 2x + 5y + 3z + s_2 &= 15 \\ x, y, z, s_1, s_2 &\geq 0 \end{aligned}$$

So our initial tableau is

$$\begin{pmatrix} 3 & 2 & 1 & 1 & 0 & 0 & 10 \\ 2 & 5 & 3 & 0 & 1 & 0 & 15 \\ -2 & -3 & -4 & 0 & 0 & 1 & 0 \end{pmatrix}$$

We now perform a pivot operation:

(**1**) we first choose the most negative value in the bottom row: -3, this will be our column.

(**2**) we then compute b_j/A_{ji} and take the row with the minimum value. The values are $10/3$ and $15/2$, so we choose 2 as the pivot.

Row reducing will give

$$\begin{pmatrix} 0 & -5.5 & -3.5 & 1 & -1.5 & 0 & -12.5 \\ 1 & 2.5 & 1.5 & 0 & 0.5 & 0 & 7.5 \\ 0 & 2 & -1 & 0 & 1 & 1 & 15 \end{pmatrix}$$

The next pivot column will be the third column, and computing the quotients gives $-12.5/-3.5 = 25/7$ and $7.5/1.5 = 5$ so we choose the second row as the pivot column. The resulting tableau is

$$\begin{pmatrix} 0 & -5.5 & -3.5 & 1 & -1.5 & 0 & -12.5 \\ 2/3 & 5/3 & 1 & 0 & 1/3 & 0 & 5 \\ 2/3 & 11/3 & 0 & 0 & 4/3 & 1 & 20 \end{pmatrix}$$

The bottom row is all positive, so we can no longer pivot. Thus the maximum value is 20. ◊

1.4 The Simplex Algorithm; Phase I

So far what we discussed relies on us having an initial bfs to our problem, otherwise our initial tableau is not in canonical form (and then we have to find a basis for which the bfs is non-negative). Instead of iterating over each basis, we can instead solve an LP to find an initial bfs.

Suppose we want to maximize $c^\top x$ constrained to $Ax = b$. Now instead let us minimize $1^\top s$ (1 is the vector containing all ones) constrained to $(A \mid I)\binom{x}{s} = b$ as well as $-c^\top x + w = 0$ for some other variable w (this is in order to keep track of the changes to $c^\top x$ as we row reduce). This is equivalent to $Ax + s = b$, i.e. $Ax \leq b$, and the idea is that if we can make $s = 0$ then the resulting bfs will be a vertex in the polytope $Ax = b$. So we minimize $1^\top s$ (equivalently, maximize $-1^\top s$) and if we get $s = 0$ then we have an initial bfs. Otherwise, we can't minimize s to 0 and so we have no solution to $Ax = b$, meaning our polytope is empty.

This procedure is the first step in the simplex algorithm, also called *Phase I*. The second step (row reducing the tableau) is called *Phase II*.

Example: suppose we want to maximize

$$2x + 3y + 4z$$

subject to

$$\begin{aligned} 3x + 2y + z &= 10 \\ 2x + 5y + 3z &= 15 \\ x, y, z &\geq 0 \end{aligned}$$

So our initial tableau is

$$\begin{pmatrix} 3 & 2 & 1 & 0 & 10 \\ 2 & 5 & 3 & 0 & 15 \\ -2 & -3 & -4 & 1 & 0 \end{pmatrix}$$

which is not in canonical form. So now according to phase I, we want to maximize $-u - v$ subject to

$$\begin{aligned} 3x + 2y + z + u &= 10 \\ 2x + 5y + 3z + v &= 15 \\ -2x - 3y - 4z + w &= 0 \\ x, y, z, u, v &\geq 0 \end{aligned}$$

The initial tableau for this is

$$\begin{pmatrix} 3 & 2 & 1 & 0 & 1 & 0 & 0 & 10 \\ 2 & 5 & 3 & 0 & 0 & 1 & 0 & 15 \\ -2 & -3 & -4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Converting to its canonical form, we get

$$\begin{pmatrix} 3 & 2 & 1 & 0 & 1 & 0 & 0 & 10 \\ 2 & 5 & 3 & 0 & 0 & 1 & 0 & 15 \\ -2 & -3 & -4 & 1 & 0 & 0 & 0 & 0 \\ -5 & -7 & -4 & 0 & 0 & 0 & 1 & -25 \end{pmatrix}$$

We could take the second column as the pivot column, but let's choose column 3 instead. We then perform the simplex algorithm on this to give an initial tableau. ◊

If some of the constraints have slack variables, we can optimize this so that we only add artificial variables to constraints without them. This way we still start with a canonical form, and we have fewer artififical variables to deal with.

Now suppose we have the linear program of maximizing $c^\top x$ subject to $Ax = b$ where b is not non-negative. Let us assume that A is in canonical form, then simply converting $R_i(A)x = b_i$ to $-R_i(A)x = -b_i$ will not work, as it will affect the canonicalness of A. For every constraint where b_i is negative, we substract some artificial variable a_0 from the constraint, so it becomes $\sum_j a_{ij}x_i - a_0 = b_i$, and we want to minimize a_0, i.e. maximize $-a_0$. We then add a_0 to the initial basis, we do this to make $\bar{b}$ nonnegative. This is because if we pivot on the ith row, then this means that b_i/A_{ij} (where j is the row corresponding to a_0) is maximal, but $A_{ij} = -1$ so $-b_i$ is maximal. This will set b_i to $-b_i$ (since remember the coefficient is -1), and for every other b_j which is negative (so a_0's coefficient is -1), this becomes $b_j - b_i \geq 0$.

This makes more sense when our constraint is $a_1x_1 + \cdots + a_nx_n \leq -b$, we add a slack variable to become $a_1x_1 + \cdots + a_nx_n + s = -b$. But then setting $x_i = 0$ gives us an infeasible solution $s = -b < 0$. So we add an artificial variable a, $a_1x_1 + \cdots + a_nx_n + s - a = -b$, then our goal is to now maximize $-a$ so that it becomes zero.

Example: we want to maximize $5x_1 - x_2 - x_3$ subject to

$$\begin{aligned} 3x_1 - x_2 - x_3 &\leq -1 \\ x_1 + 2x_2 - x_3 &\leq -2 \\ 2x_1 + x_2 &\leq 2 \\ x_1 + x_2 &= 1 \\ x_1, x_2, x_3 &\leq 0 \end{aligned}$$

We need two artificial variables: one for the equality constraint, and one for the negative constraints. We also add slack variables. So our problem becomes maximizing $-a_1 - a_2$ subject to

$$\begin{aligned} 3x_1 - x_2 - x_3 + s_1 - a_1 &= -1 \\ x_1 + 2x_2 - x_3 + s_2 - a_1 &= -2 \\ 2x_1 + x_2 + s_3 &= 2 \\ x_1 + x_2 + a_2 &= 1 \\ -5x_1 + x_2 + x_3 + z &= 0 \end{aligned}$$

So the initial tableau is

$$\left(\begin{array}{cccccccccccc} x_1 & x_2 & x_3 & s_1 & s_2 & s_3 & z & a_1 & a_2 & w & b \\ \hline 3 & -1 & -1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & -1 \\ 1 & 2 & -1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & -2 \\ 2 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 2 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ -5 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right)$$

To make this canonical, we must adjust the objective row so that a_2 is in the basis:

$$\left(\begin{array}{cccccccccccc} x_1 & x_2 & x_3 & s_1 & s_2 & s_3 & z & a_1 & a_2 & w & b \\ \hline 3 & -1 & -1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & -1 \\ 1 & 2 & -1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & -2 \\ 2 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 2 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ -5 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline -1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 \end{array}\right)$$

Now we pivot with respect to a_0, which enters, and s_2 leaves. We then continue normally until we get

$$\left(\begin{array}{cccccccccccc} x_1 & x_2 & x_3 & s_1 & s_2 & s_3 & z & a_1 & a_2 & w & b \\ \hline 1 & 0 & 0 & 1/5 & -1/5 & 0 & 0 & 3/5 & 0 & 0 & 4/5 \\ 0 & 0 & 1 & -1/5 & -4/5 & 0 & 0 & 7/5 & 1 & 0 & 16/5 \\ 0 & 0 & 0 & -1/5 & 1/5 & 1 & 0 & -8/5 & 0 & 0 & 1/5 \\ 0 & 1 & 0 & -1/5 & 1/5 & 0 & 0 & 2/5 & 0 & 0 & 1/5 \\ 0 & 0 & 0 & 7/5 & -2/5 & 0 & 1 & 6/5 & -1 & 0 & 3/5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right)$$

Phase I is thus successful since we have maximized to 0. We can then perform phase II on the canonical tableau

$$\left(\begin{array}{cccccccc} x_1 & x_2 & x_3 & s_1 & s_2 & s_3 & z & b \\ \hline 1 & 0 & 0 & 1/5 & -1/5 & 0 & 0 & 4/5 \\ 0 & 0 & 1 & -1/5 & -4/5 & 0 & 0 & 16/5 \\ 0 & 0 & 0 & -1/5 & 1/5 & 1 & 0 & 1/5 \\ 0 & 1 & 0 & -1/5 & 1/5 & 0 & 0 & 1/5 \\ \hline 0 & 0 & 0 & 7/5 & -2/5 & 0 & 1 & 3/5 \end{array}\right)$$

Now s_2 enters and s_3 leaves since the most negative value in the bottom row is $-2/5$ and the largest quotient is 1. This gives

$$\left(\begin{array}{cccccccc} x_1 & x_2 & x_3 & s_1 & s_2 & s_3 & z & b \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 4 & 0 & 4 \\ 0 & 0 & 0 & -1 & 1 & 5 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 2 & 1 & 1 \end{array}\right)$$

So the maximum value is 1 with $x_1 = 1$, $x_2 = 0$ and $x_3 = 4$. ◊

1.5 Duality

1.5.1 Definition

Call an LP problem of the form: maximize $c^\top x$ such that $Ax \leq b, x \geq 0$ **primal**. Its **dual** is: minimize $b^\top y$ such that $A^\top y \geq c, y \geq 0$.

1.5.2 Lemma (The Weak Duality Theorem)

If x is a feasible solution to the primal LP and y to the dual LP, then $c^\top x \leq b^\top y$.

Proof: this is just a chain of inequalities:

$$c^\top x = x^\top c \leq x^\top (A^\top y) = (x^\top A^\top) y = (Ax)^\top y \leq b^\top y$$ ■

1.5.3 Theorem (The Strong Duality Theorem)

If x is a maximal solution to the primal LP, and y a minimal solution to the dual LP, then $c^\top x = b^\top y$.

Proof: let us show that if x is a maximal solution to the primal problem, then there exists a solution y to the dual problem such that $c^\top x = b^\top y$. By the weak duality theorem, this is sufficient.

Suppose first that our problem is in slack form: $Ax = b$. Let x be a maximal solution, and its corresponding basis is B. Suppose that $A = A_B \,|_B\, A_N$ and $c^\top = c_B^\top \,|_B\, c_N^\top$, then our initial tableau, multiplied by $A_B^{-1} \oplus (1)$ is

$$\begin{pmatrix} A_B^{-1} & \\ & 1 \end{pmatrix} \cdot \begin{pmatrix} A & 0 & b \\ -c^\top & 1 & 0 \end{pmatrix} = \begin{pmatrix} I & |_B & A_B^{-1}A_N & 0 & A_B^{-1}b \\ -c_B^\top & |_B & -c_N^\top & 1 & 0 \end{pmatrix}$$

Row reducing gives us

$$\begin{pmatrix} I & |_B & A_B^{-1}A_N & 0 & A_B^{-1}b \\ 0^\top & |_B & -c_N^\top + c_B^\top A_B^{-1} A_N & 1 & z_B \end{pmatrix}$$

Since x is optimal, the bottom row must be positive, so

$$c_N^\top - c_B^\top A_B^{-1} A_N \leq 0$$

but trivially, we also know (since the following is equal to 0)

$$c_B^\top - c_B^\top A_B^{-1} A_B \leq 0$$

Thus by concatenating them with their order determined by B,

$$c^\top - c_B^\top A_B^{-1} A \leq 0$$

Now, our problem was $Ax \leq b$, which is equivalent to $(A \mid I)\binom{x}{s} = b$ by introducing slack variables. And the objective function is $(c^\top \mid 0)\binom{x}{s}$, so by our above computation (by splitting according to which indexes are slack variables), we get

$$\begin{aligned} \text{For non-slack variables:} \quad & c^\top - c_B^\top \bar{A}_B^{-1} A \leq 0 \\ \text{For slack variables:} \quad & 0^\top - c_B^\top \bar{A}_B^{-1} I \leq 0 \end{aligned}$$

where $\bar{A} = A \mid I$. So let us define $y^\top = c_B^\top \bar{A}_B^{-1}$, then we have that

$$c^\top - c_B^\top \bar{A}_B^{-1} A = c^\top - y^\top A \leq 0 \implies y^\top A \geq c^\top$$

and $y^\top \geq 0$. So y is a solution to the dual LP problem. Furthermore

$$b^\top y = y^\top b = c_B^\top \bar{A}_B^{-1} b = c_B^\top x_B = c^\top x$$

as required. ■

2 Approximation Algorithms

2.1 Optimization Complexity Classes

2.1.1 Definition

An **optimization problem** is a tuple $\mathcal{O} = (I, \mathsf{Sol}, m, \mathsf{type})$, where

(**1**) I is a set called the **instance set**, whose elements are instances of the problem.

(**2**) Sol is a function such that for every instance $i \in I$, $\mathsf{Sol}(i)$ is the set of all feasible solutions to the instance i.

(**3**) m is a function such that for every instance $i \in I$ and solution $s \in \mathsf{Sol}(i)$, $m(i, s)$ is the **measure** of the solution.

(**4**) $\mathsf{type} \in \{\max, \min\}$ which tells us if we want to maximize or minimize the measure of a solution.

Then the **optimal measure** of an instance $i \in I$ is

$$\mathrm{opt}(i) = \underset{s \in \mathsf{Sol}(i)}{\mathsf{type}}\ m(i, s)$$

For example, for the problem of finding the shortest path between two vertices s and t can be described as such:

- $I = \{(G = (V, E), s, t) \mid s, t \in V\}$
- $\mathsf{Sol}(G, s, t) = \{P \mid P \text{ is a path from } s \text{ to } t \text{ in } G\}$
- $m((G, s, t), P) = |P|$
- $\mathsf{type} = \min$

Given an optimization problem $\mathcal{O}$, its corresponding decision problem $\mathcal{D}_\mathcal{O}$ is the problem of determining if given a $k \in \mathbb{R}$ and instance $i \in I$, whether there exists a solution $s \in \mathsf{Sol}(i)$ such that $m(i, s) \geq k$ for $\mathsf{type} = \max$ and $m(i, s) \leq k$ for $\mathsf{type} = \min$.

2.1.2 Definition

The complexity class **NPO** (**NP** optimization) is the class of all optimization problems $\mathcal{O}$ such that its corresponding decision problem $\mathcal{D}_\mathcal{O}$ is in **NP**.

Call an optimization problem $\mathcal{O}$ **NPO**-complete if $\mathcal{D}_\mathcal{O}$ is **NP**-complete.

2.1.3 Definition

An **NPO** problem $\mathcal{O}$ is in **PO** (polynomial optimization) if there is a deterministic polynomial-time algorithm to compute $\mathrm{opt}(i)$.

Since NPO-complete problems are NPO-complete, we currently (and maybe always) do not have an efficient method of solving them. So the next best thing is approximating solutions.

2.2 Approximation Algorithms

2.2.1 Problem

Given a graph $G = (V, E)$, a set $S \subseteq V$ is a **vertex cover** if for every edge $\{u, v\}$, either $u \in S$ or $v \in S$. The optimization problem of vertex covers is to find the size of the minimum vertex cover of an input

graph.

As you should know, the decision problem is NP-complete. So the optimization problem is NPO-complete.

We can come up with a naive algoritjm to solve this:

```
1. function VERTEX-COVER(G = (V, E))
2.     A ← ∅
3.     while (there exists an edge {u, v} st u, v ∉ A)
4.         A ← A ∪ {u, v}
5.     end while
6.     return A
7. end function
```

2.2.2 Theorem

The above algorithm is a 2-approximation, that is A is a vertex cover and $|A| \leq 2 \cdot |\text{Opt}|$.

Proof: let M be the set of edges chosen in the algorithm. Then Opt must include a vertex from each edge in M as it is a vertex cover, so $|M| \leq |\text{Opt}|$. Furthermore, $|A| = 2|M|$, so

$$|A| = 2|M| \leq 2|\text{Opt}|$$

as required. ■

This analysis is tight: there exists a graph such that $|A| = 2|\text{Opt}|$.

2.2.3 Definition

APX (approximation) is the family of all approximation problems $\mathcal{O} \in$ **NPO** such that for some constant $c \geq 0$, there is a deterministic polynomial-time algorithm such that for every $i \in I$, it finds a solution $s \in \mathsf{Sol}$ such that

(**1**) If type = min then $c \geq 1$ and $m(i, s) \leq c\,\text{opt}(i)$.

(**2**) If type = max then $c \leq 1$ and $m(i, s) \geq c\,\text{opt}(i)$.

So Vertex-Cover is in **APX**.

2.2.4 Problem

Given a graph $G = (V, E)$ the problem of **max cut** is the problem of finding a cut $S \subseteq V$ that maximizes the number of edges in the cut

$$E(S) = E(S, V \setminus S) = \{\{u, v\} \in E \mid u \in S, v \notin S\}$$

The decision problem of max cut is NP-complete, so max cut is NPO-complete. We describe an algorithm to approximate max cut:

```
1. function MAX-CUT(G = (V, E))
2.     S ← ∅
3.     while (true)
4.         if (there exists a v ∉ S such that |E(S ∪ {v})| > |E(S)|)
5.             S ← S ∪ {v}
6.         else if (there exists a v ∈ S such that |E(S \ {v})| > |E(S)|)
7.             S ← S \ {v}
8.         else
9.             return S
10.        end if
11.    end while
```

12. **end function**

There are at most $\binom{n}{2}$ edges in the graph, and thus the maximum possible size of $E(S)$ is $\binom{n}{2}$. Notice that at each iteration, $|E(S)|$ increases by at least 1, and so there can be at most $\binom{n}{2}$ iterations, so this algorithm is polynomial-time.

Notice that if the algorithm returns S, then for $v \in S$, $|E(\{v\}, V \setminus S)| \geq \frac{\deg v}{2}$ since otherwise more than half of v's neighbors are in S and then the algorithm would've placed v in $V \setminus S$. Similarly for $v \notin S$, $|E(\{v\}, S)| \geq \frac{\deg v}{2}$. Thus

$$|E(S, V \setminus S)| \geq \frac{1}{2} \sum_{v \in V} \frac{\deg v}{2} = \frac{|E|}{2} \geq \frac{\text{opt}}{2}$$

This is because when we sum over $v \in V$, we count how many edges are in the cut, but since we count both $v \in S$ and $v \notin S$, we double count. So we must divide by 2.

Thus this algorithm is a $\frac{1}{2}$-approximation, meaning max cut is in APX.