# Advanced Algorithms

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

## Contents

**1 Linear Programming** **1**

# 1 Linear Programming

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).

**1.0.1 Definition**

A **Linear Programming** problem is the problem of maximizing some 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 $f(x_1, \ldots, x_n \,;\, -c_1, \ldots, -c_n)$.

**1.0.2 Definition**

An LP problem 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 problem can be converted to an equivalent LP problem in standard form. Suppose we have the LP problem $f(x_1, \ldots, x_n \,;\, c_1, \ldots, c_n)$ 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

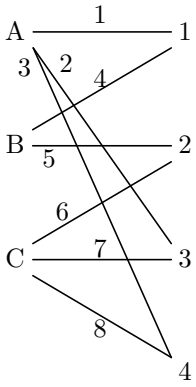
$$\sum_{j=1}^{n} \alpha_{i,j} (x_i^+ - x_i^-) \leq \beta_i \qquad (1 \leq i \leq m)$$

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 problem in standard form, we can write it as a problem of the form

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

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 $i$th 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 $\text{VC}_{opt}$ (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. ◊

**1.0.3 Definition**

The **slack form** of an LP problem 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 problem 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.

In slack form, we can just set all $x_j = 0$ and $s_i = \beta_i$, and this gives us a solution (though it is not maximal), and is called the basic feasible solution (bfs).

**Example:** Returning to our pizzeria example, but in slack form, we want to maximize $25x_1 + 30x_2 + 40x_3$ under the constraints

$$\begin{aligned} 2x_1 + x_2 + x_3 + s_1 &= 1000 \\ 2x_1 + 3x_2 + 2x_3 + s_2 &= 500 \\ x_1 + 2x_2 + 3x_3 + s_3 &= 800 \end{aligned}$$

We can rewrite this as (swapping $s_i$ with $x_{i+3}$):

$$\begin{aligned} x_4 &= 1000 - 2x_1 - x_2 - x_3 \\ x_5 &= 500 - 2x_1 - 3x_2 - 2x_3 \\ x_6 &= 800 - x_1 - 2x_2 - 3x_3 \end{aligned}$$

The basic variables are the variables on the left. Now, we start at the bfs $(0, 0, 0, 1000, 500, 800)$. We want to maximize $25x_1 + 30x_2 + 40x_3$, so let us maximize $x_3$ (with the largest coefficient) using the smallest constraint. We call this variable the *pivot*:

$$x_3 = 250 - x_1 - 1.5x_2 - 0.5x_5$$

so if we set $x_5 = 0$ we get the bfs (since now that $x_3$ is on the left, it is a basic variable), $(0, 0, 250, 1000, 0, 800)$.

Now since basic variables cannot be in our expression we are maximizing, we write it as

$$25x_1 + 30x_2 + 40(250 - x_1 - 1.5x_2 - 0.5x_5) = 10{,}000 - 15x_1 - 30x_2 - 20x_5$$

And

$$\begin{aligned} x_4 &= 1000 - 2x_1 - x_2 - (250 - x_1 - 1.5x_2 - 0.5x_5) \\ x_3 &= 250 - x_1 - 1.5x_2 - 0.5x_5 \\ x_6 &= 800 - x_1 - 2x_2 - 3(250 - x_1 - 1.5x_2 - 0.5x_5) \end{aligned}$$

Since in the expression we are maximizing all the coefficients are negative, we have reached a bfs which gives us a maximum value: 10,000. ◊

This algorithm is called the *simplex algorithm.*

**Example:** suppose we want to maximize $x_1 + x_2 + x_3$ under the constraints $x_1 + x_2 \leq 10$ and $x_2 \geq x_3$. In slack form we have

$$x_4 = 10 - x_1 - x_2, \qquad x_5 = x_2 - x_3$$

This has a bfs of $(0, 0, 0, 10, 0)$. Choosing $x_1$ as the pivot we get

$$x_1 = 10 - x_2 - x_4, \qquad x_5 = x_2 - x_3$$

and so we maximize $10 - x_4 + x_3$. So we now choose $x_3$ as a pivot, and we get

$$x_1 = 10 - x_2 - x_4, \qquad x_3 = x_2 - x_5$$

and we now must maximize $10 - x_4 + x_2 - x_5$, and so we now choose $x_2$ as a pivot:

$$x_2 = 10 - x_1 - x_4, \qquad x_3 = 10 - x_1 - x_4 - x_5$$

and so we must maximize $10 - x_3 + 10 - x_1 - x_4 - x_5$. So the maximum value is 20 with bfs $(0, 10, 10, 0, 0)$. ◊

This example highlights an important point: what if through choosing a pivot we get back to the original list of basic variables, or a list of basic variables we have already encountered? Then we have a loop, and our algorithm doesn't terminate.

Say our algorithm goes through bfs-s like so:

$$\text{bfs}_1 \to \text{bfs}_2 \to \cdots \to \text{bfs}_\ell$$

and it doesn't have a loop. Then we have $\binom{n+m}{m}$ choices for basic variables (since we choose which $m$ of the $n + m$ variables are basic), and so we must have $\ell \leq \binom{n+m}{m} = \binom{n+m}{n}$.

Now suppose we have a standard form LP problem: we maximize $\vec{c}^\top \vec{x}$ with the constraints $A\vec{x} \leq \vec{b}$ and $\vec{x} \geq 0$. Now if we look at

$$(y_1, \ldots, y_m)A\vec{x} = \sum_{i=1}^{n} \left( \sum_{j=1}^{m} \alpha_{ji} y_j \right) x_i$$

If we get that $\sum_{j=1}^{m} \alpha_{ji} y_j \geq c_i$, i.e. $A^\top \vec{y} \geq \vec{c}$, then we have that

$$\vec{y}^\top \vec{b} \geq \vec{y}^\top A\vec{x} \geq \sum_{i=1}^{n} c_i x_i = \vec{c}^\top \vec{x}$$

So we have that there exists some duality between the two problems:

$$\begin{array}{lll} \text{Maximize} & \vec{c}^\top \cdot \vec{x} & \\ \text{Such that} & A\vec{x} \leq \vec{b} & \iff \\ & \vec{x} \geq \vec{0} & \end{array} \qquad \begin{array}{lll} \text{Minimize} & \vec{b}^\top \cdot \vec{y} \\ \text{Such that} & A^\top \vec{y} \leq \vec{c} \\ & \vec{y} \geq \vec{0} \end{array}$$