Linear Programming[1]

In this lecture, my goal is to tell you a very large class of problems which can be solved efficiently in polynomial time. In my opinion, this is the crown jewel in the algorithmic treasure chest. Unfortunately, the algorithm is not simple and takes some time to explain. Nevertheless, I think all of you should *know* about this class of problems so that if, in the future, you can fit your problem into this class, you can then look up the algorithm itself. Moving to the problem itself.

1 Linear Programming

Let us recall some linear algebra. Suppose A is an $n \times n$ matrix, and $b \in \mathbb{R}^n$ is a column vector. We have all probably see a system of linear equations $Ax = b$. If A is *invertible* (equivalently, determinant is non-zero), then there is only one such x, namely $x = A^{-1}b$. If A were an $m \times n$ matrix with $m \leq n$, then there can be multiple such x's. In fact, the set of all $\{x : Ax = b\}$ forms a *vector space*.

Remark: *Given A how does one find A^{-1}? This, after all, is also an algorithm. More generally, solving $Ax = b$ is done via an algorithm. One such algorithm transforms the matrix into "row-echelon" form, or "upper-triangular" form, and then the solution is easy. Naively, this can be done in $O(n^3)$ arithmetic operations[a]. Do you recall this?*

[a] One has to be careful here. If any two numbers can be added/divided in $O(1)$ time, then we get $O(n^3)$ algorithm. But the numbers when you multiply them can get rather large especially if you multiply them n times. Nevertheless, there is an $O(n^3)$ time algorithm, but that is not a trivial implementation.

The fun begins when we add *inequalities* in the mix. Suppose we are interested in the question whether or not there is an x such that $Ax = b$ *and* $x \geq 0$, that is, all coordinates of x are non-negative. Suddenly the problem is not so easy any more. Of course, if A is $n \times n$ and full-rank (that is A^{-1} exists) then of course the problem doesn't even change: there is only one x such that $Ax = b$ and if that x is not all non-negative, then the answer is NO. However, when A is $m \times n$ (or rank-deficient), then the question is asking whether the vector space $\{x : Ax = b\}$ has a all non-negative solution. This is the fundamental question in linear programming.

LINEAR PROGRAMMING: FEASIBILITY VERSION
Input: An $m \times n$ constraint matrix A, an m-dimensional vector $b \in \mathbb{R}^m$
Output: $x \in \mathbb{R}^n$ such that $Ax = b$ and $x_i \geq 0$ for all i, or assert NO if no such vector exists.
Size: The number of bits required to describe the input (A, b).

Traditionally, the term linear programming is used for the *optimization* version of the problem. In this problem, along with A and b, one is given a cost vector $c \in \mathbb{R}^n$. The objective is to either say NO if there is no $x \geq 0$ satisfying $Ax = b$, or to return an x^* which *minimizes* $\langle c, x \rangle$ over all $\{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$. Here $\langle c, x \rangle$ is the shorthand for the dot product $\sum_{i=1}^{n} c_i x_i$.

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 19th Mar, 2022*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

LINEAR PROGRAMMING : OPTIMIZATION VERSION

Input: $m \times n$ constraint matrix A, an m dimensional constraint vector b, and an n-dimensional cost vector c.

Output: An n-dimensional vector x which is a solution to

$$\begin{aligned} \min \quad & \langle c, x \rangle \\ & Ax = b \\ & x \geq 0 \end{aligned} \tag{1}$$

or assert there is no solution to $\{x : Ax = b, x \geq 0\}$.

Size: The number of bits required to describe all the data.

Remark: *Note that the entries of A, b, c can be arbitrary rational numbers. In particular, we can have* $\max$ *replace the* $\min$ *and replace some of the* ≥ 0 *(of the n of them) with* ≤ 0 *and the problem remains the same. This can be done by suitable negating. For instance,* $\max\langle c, x\rangle$ *is the same as* $\min\langle -c, x\rangle$*. If $x_i \leq 0$, then this is identical to having $x_i \geq 0$ and negating the ith column of A.*

One may also see linear programs which have *only* inequalities. Indeed, depending on your application this version may capture your problem better. I am adding it just in case.

LINEAR PROGRAMMING : OPTIMIZATION VERSION "PART II"

Input: $m \times n$ constraint matrix A, an m dimensional constraint vector b, and an n-dimensional cost vector c.

Output: An n-dimensional vector x which is a solution to

$$\begin{aligned} \min \quad & \langle c, x \rangle \\ & Ax \geq b \end{aligned}$$

or assert there is no solution to $\{x : Ax \geq b\}$.

Size: The number of bits required to describe all the data.

There is really *no* difference between the two versions. That is, if we can solve the first version, then we can solve the second version. Here's how. We introduce *slack variables* $s_1, \ldots, s_m$. If we let s be an m-dimensional vector and assert[2] $\{(x, s) : Ax - I_m s = b, (x, s) \geq 0\}$, then looking at this solution and taking just the "x-part" gives a solution where $Ax \geq b, x \geq 0$. Similarly, given any solution x satisfying $Ax \geq b, x \geq 0$, we can find $s \in \mathbb{R}^m$ such that $s \geq 0$ and $Ax - Is = b$. Indeed, the most general linear program looks like this.

LINEAR PROGRAMMING : "GENERAL VERSION"

Input: $m \times n$ inequality constraint matrix A with m dimensional constraint vector $b_{\geq}$, $k \times n$ equality constraint matrix B with k dimensional constraint vector $b_{=}$, and an n-dimensional cost vector c.

[2]Here I_m is the $m \times m$ identity matrix

Output: An n-dimensional vector x which is a solution to

$$\begin{aligned} \min \quad & \langle c, x \rangle \\ & Ax \geq b_{\geq} \\ & Bx = b_{=} \end{aligned}$$

or assert there is no solution to the above constraints.
Size: The number of bits required to describe all the data.

Here is the crown jewel.

Theorem 1 (Dantzig, Shor, Nemirovskii, Khachiyan, Karmarkar,...)**.** There is a polynomial time algorithm for linear programming.

This, by any means, is not a trivial theorem. It is one of the deepest facts uncovered in the last 50 years.

1.1 Some Examples

Maximum Flows. We now see that the maximum flow problem is a simple special case of linear programming. To see this, we need to describe the constraint matrices, constraint vector, and the cost vector.

Consider the *signed* edge-incidence matrix. It is an $|V| \times |E|$ matrix. The rows correspond to the vertices. The columns correspond to the edges. For every edge (u, v), the column corresponding to this edge has two non-zero entries: it has a -1 in the uth row and a $+1$ in the vth row, and 0 everywhere else. Note that given any x_e for all edges $e \in E$¡ we get that $B \cdot x$ is an $|V|$-dimensional vector whose uth coordinate corresponds to precisely $\sum_{(w,u)\in E} x_{wu} - \sum_{(u,v)\in E} x_{uv}$. This is simply the excess $\mathsf{excess}_x(u)$. Now, we know what the equality constraint matrix and equality constraint vector is. The matrix is simply the rows of B except the row corresponding to s and t are deleted. The equality vector is the all 0s vector. This captures the conservation constraints: $\mathsf{excess}_x(v) = 0$ for all $v \notin \{s, t\}$.

The capacity constraints are easily captured by the $|E|$-dimensional identity matrix with the constraint vector being the vector of the capacities.

The cost vector is simply -1 on all edges of the form (w, t) and $+1$ for edges of the form (t, u). This, $\langle c, x \rangle$ measures $-\mathsf{excess}_x(t)$ and since we want to *maximize* the value of the flow, that is, $\mathsf{excess}_x(t)$, this is equivalent to minimizing $\langle c, x \rangle$.

Thus, the maximum flow problem is a special case of linear programming. But we can now see something more. We can see that, in fact, put in *any* cost vector and get special kinds of flow problems. For example, suppose every edge $e \in E$ had a *cost* $c(e)$ (as well as its capacity) which indicated the cost of sending one unit of flow through it. One could ask to find a maximum flow of minimum possible cost. Well, one can first solve the maximum flow to find the value F^* of the maximum flow. And then, one can just *add it as a constraint*. The simplest way is to just let the equality constraint matrix be the whole of the signed edge-incidence matrix with the constraint vector corresponding to s and t have values $-F^*$ and $+F^*$. Now, just minimizing $\langle c, x \rangle$ satisfying these new constraints would give the ***minimim cost maximum flow***. Just like maximum flows had many applications, this also has many applications. For example, given two vertices s and t, one can find k-edge disjoint paths (this was our first application of flow) such that the total cost is minimized (and this is where we use minimum cost flows).

Remark: *"Hold on!", I hear you cry. The disjoint path application only makes sense if the flow on any edge is either* 0 *or* 1*. And indeed, one thing that the Ford-Fulkerson algorithm assured is that if the capacities are integer valued then there is an optimum solution which is also integer valued. Is that true for linear programs? Is the solution to a linear program integer valued if the constraint vector* b *is integer valued? In general, no. However, for certain* constraint matrices *the answer is yes. In particular, if the matrix is* ***totally unimodular*** *(TU), that is, if the determinant of any square sub-matrix of the constraint matrices (equality and inequality both taken together) is in the set* $\{-1, 0, +1\}$*, then for any integer-valued constraint vector* b *the solution to the linear program is integer valued. This is true* even if *the cost vector is not integer valued. The reason for this is not hard and stems from the fact that if the determinant of an integer square matrix is* ± 1*, then the inverse is also an integer matrix.*

Now, the signed edge-incidence matrix is indeed TU. This is not hard to show, and I leave this as an exercise to the interested reader.

Plane Fitting. Let's do another example, one which we haven't seen in class. Let us do an example from machine learning. Suppose we are given *data points* (x_i, y_i) for $1 \leq i \leq m$. Here, every $x_i \in \mathbb{R}^n$ is an n-dimensional point, while the $y_i \in \mathbb{R}$'s are scalar labels. Ideally, these labels are some *affine* measurements on the data point. That is, there is some $a \in \mathbb{R}^n, b \in \mathbb{R}$, and all (x_i, y_i), in an ideal world, should lie in the set $\{(x, y) \ : \ y = \langle a, x\rangle + b\}$.

However, there is noise/inaccuracies in data collection, and so the objective is to find the *closest* affine subspace to the obtained data. What does closest mean? Given any affine subspace $\{(x, y) \ : \ y = \langle a, x\rangle + b\}$, let us denote the *error* of (x_i, y_i) as $|y_i - (\langle a, x_i\rangle + b_i)\,|$. That is, the absolute value[3] of the deviation of y_i from what the affine function suggests. Question is: given the data point, can we find the best (a, b)?

Mathematically, the problem is to find $a \in \mathbb{R}^n, b \in \mathbb{R}$ such that $\sum_{i=1}^{m} |y_i - (\langle a, x_i\rangle + b_i)|$ is minimized. I claim that this is a linear program. At first glance, you may think there are *no* constraints at all. You are correct at one level, but note that the objective function is not of the form $\langle , \rangle$, that is, it is not a linear function of the variables (a, b). One needs to do a little work to cast it as a linear program.

The idea is to introduce new variables z_i, $1 \leq i \leq m$ which is supposed to capture the absolute value. This is done by asserting that z_i is greater than what is inside the absolute value, and also greater than the negative of what is inside the absolute value. This leads to the constraints. Let me write the linear program in a form that one usually writes linear programs in, and then elucidate what the matrices are.

$$\begin{aligned} \min \quad & \sum_{i=1}^{m} z_i & \\ & z_i + \langle a, x_i\rangle + b \ \geq \ \ y_i, & \forall 1 \leq i \leq m \\ & z_i - \langle a, x_i\rangle - b \ \geq \ -y_i, & \forall 1 \leq i \leq m \end{aligned}$$

Note that I have not described the matrix. But one can easily tease it out. There is an inequality constraint matrix. It has $2m$ rows, with 2 inequalities corresponding to the ith data point. There are $m+n+1$ columns. The first m columns correspond to z_i's, the next $n+1$ to the variables (a, b). One of the two rows corr. to ith data point has 1 in the z_ith column, followed by $(x_i, +1)$ and the constraint vector has y_i. The other row has $(-x_i - 1)$ and the constraint vector has value $-y_i$.

[3] In linear regression, which is the problem you probably have studied in a machine learning course, one usually takes the square of this absolute value as the measure. There are technical and statistical reasons for that. As a computation problem, they become slightly different