COMS 4995-2: Advanced Algorithms (Spring'20)	Feb 27, 2020

Lecture 12: Optimization / Linear Programming

Instructor: *Alex Andoni* Scribes: *John Daciuk and Sagar Lal*

1 Introduction

General LP Objective: $minf(x)$; $x \in F \subset \mathbb{R}$ where F is the set of feasible solutions.

There are many types of problems that are included in linear programming. When a problem can be put in terms of an objective function and competing constraints, it might be solved with LP. Although we can't solve sorting with LP, if you go to a desert island and can only take one algorithm with you, take an LP solver[1]. Here is an example LP problem:

Example 1: Min-conductance of G
$\phi(G) = \min_{s \notin \emptyset, vol(s) \leq \frac{1}{2} vol(v)} \frac{|\partial S|}{vol(s)}$ where $vol(s) = \sum_{i \in s} d$

This problem can be translated to the following problem:

Obj: unknowns is $x \in \mathbb{R}^n \Rightarrow$ indicators $\mathbb{1}s \Rightarrow \begin{cases} 1, & \text{if } i \in s \\ 0, & \text{if } i \notin s \end{cases}$

$min \frac{x^T L x}{\sum_{i \in [a]} x_i \cdot d_i}$ Note L is the laplacian, minimize such that the following conditions hold:

1. $x \in 0, 1 \Leftrightarrow x_i(1 - x_i) = 0$
2. $\sum x_i \geq 1$
3. $\sum x_i \cdot d_i \leq \frac{1}{2} \sum d_i$

In effect the numerator in the minimization represents the boundary of S and the denominator represents the volume of S if the constraints hold. As we saw in last class, solving the former problem is NP-hard, so the LP version will be as well. Some intuition why this is the case is that conditions 2 and 3 are linear inequalities, but condition 1 has a polynomial of degree 2, which in general makes the problem quite difficult.

[1] Joke from lecture

1.1 Linear Programming General Form

Linear programming is a subset of optimization problems that have the following form:

$minf(x) = \sum_{i=1}^{n} c_i x_i$ s.t $Ax \geq b$ where A is a m x m matrix and $b \in \mathbb{R}^n$.

This effectively is a summarization of a series of linear inequalities of the form: $a_{1,1}x_1 + a_{1,2}x_2, , a_{1,n}x_n \geq b_1$.

Remarks:

1. $maxf(x) = -min[-f(x)]$
2. $A_1 x \geq b \Leftrightarrow (-A_1)x \leq -b_1$
3. $A_1 x = b_1 \Leftrightarrow A_1 x \leq b_1 \;\; \& \;\; A_1 x \geq b_1$

So we can already see how the same problem can be written in different ways. The general goal of the chapter is to be able to solve equations of this form via an algorithm that works in polynomial time.

Example 2: Max flow in G
Problem: Given G, capacity k; s,t $\in$ v, find $f \in \mathbb{R}^n$ that $max|f|$ s.t. f represents $s \to t$ flow.

LP Formulation:
$max \sum_{(s,u)\in E} f_{s,u} - \sum_{(u,s)\in E} f_{u,s} = |f|$ s.t.

1. f obeys capacity constraints: $0 \leq f_{u,v} \leq k_{u,v} \forall (u,v) \in E$
2. f obeys flow conservation: $\forall u \notin s,t$: $\sum_{(v,u)\in E} f_{v,u} - \sum_{(u,v)\in E} f_{u,v} = 0$

Although we can put a max flow problem into an LP solver, that doesn't mean that we'll get as good of a runtime as some of the other algorithms we discussed.

2 Standard Form of LP

One of the purposes of today's lecture is to understand the structure of the problem we're trying to solve. Once we sufficiently understand the structure, then we can design algorithms in later lectures to exploit it. Although not really mathematically distinct, we have another way of expressing an LP problem that can be more convenient. It's nothing more than a rewriting of what we saw earlier, but this will later become useful. This transformation is analogous to what we did going from the adjacency matrix to the Laplacian, which had some nice properties.
Definition 1. *Minimize $c^T x$ where $c \in \mathbb{R}^n$, such that $Ax = b$ and $x \geq 0$.*

At first glance this may appear easier than what we saw earlier. We're just looking at solutions to a system of linear equations where x is positive. However, it turns out that the two problems are equivalent.

Reduction from general form to standard form:

We have two things to address. First we are now allowing only positive variables, and second, now $Ax = b$ is an equality. The reduction proceeds in two steps.

1. To accomodate the desire for all non-negative variables, for all variables $x_i \in \mathbb{R}$ in the general form, we create two new variables x_i^+ and x_i^- with the definition $x_i \triangleq x_i^+ - x_i^-$ and $x_i^+,\ x_i^- \geq 0$. Since any real number can be written as the difference of two other positive real numbers, this is a safe step to take.

2. Plugging this into the general form would imply $A_i x \geq b \Rightarrow A_i(x_i^+ - x_i^-) \geq b$, which we want now to be an equality. If we need $A_i(x_i^+ - x_i^-) \geq b$ this is equivalent to saying we need $A_i(x_i^+ - x_i^-) - \xi_i = b$ for some non-negative ξ_i. This motivates the introduction of slack variables $\xi_i \triangleq A_i(x_i^+ - x_i^-) - b$ where we constrain $\xi_i \geq 0$.

The slack variables tell us how far off we are from the previous inequalities in general form. If $\xi_i = 0$, we call the i^{th} constraint tight. We have introduced a bunch of equations which will be captured in the new $Ax = b$ [2]. Note that our x, A and b in $Ax = b$ will not be the same as were for the general form, otherwise we could easily see that the mathematical equivalence breaks down. Rather these objects will grow in dimension. In particular x will now contain x_i^+, x_i^-, and ξ_i. The c in the objective function may also grow, and in general the objective function will not look the same. However, the key point is that we will guarantee that an optimal solution to the new linear program will yield an optimal solution to the original. More specifically, for each feasible solution x in the general form with objective value v, there is a corresponding feasible solution x' in standard form with the same objective value v and vice versa.

3 Structure of Feasible Solutions for LP

Definition 2. *x is a feasible solution if it satisfies the constraints ($Ax \geq b$ for general form). $F \triangleq$ set of feasible x.*

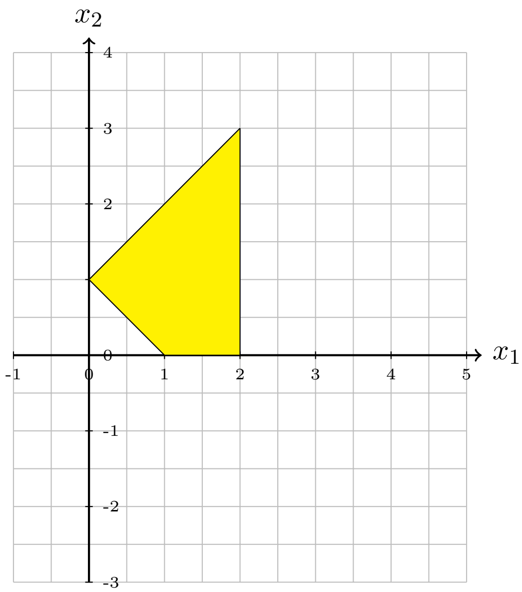
We proceed to get some intuition for what F looks like, meanwhile ignoring the objective for now. First, recall we need to satisfy $A_i x \geq b_i,\ \forall_i$. To break this down we can think about $A_1 \cdot x \geq b_1$ where A_1 is a vector of the first row of A and b_1 is just a real number. For 2 dimensional x the solutions lie on one side of a line which A_1 is perpendicular to, for 3 dimensions the solutions are confined to one side of a plane, and in general the solutions will be a half-space on one side of a hyperplane with A_1 normal to the plane. To see why A_1 is normal, consider the simplified $A_1 x \geq 0$. We've defined a hyperplane through the origin and if x sits on the hyperplane, we should get 0, thus A_1 would need to be perpendicular to x and the hyperplane.

Because we must satisfy not just $A_1 x \geq b_1$ but all $A_i x \geq b_i$, F will have to sit inside all of these half-spaces or a polytope.

Definition 3. *F is bounded if contained in a finite polytope and unbounded otherwise.*

[2] Except for the non-negativity constraint on the variables which may be implicitly taken care of.

3.1 Example of F in two dimensions



$$x_1 \leq 2$$
$$x_1 + x_2 \geq 1$$
$$x_1 - x_2 \geq -1$$
$$x_2 \geq 0$$

4 How to Find an Optimum Solution

4.1 Possibilities

Recall we need to minimize $c^T x$ such that $x \in F$. There are 3 possibilities for what can happen:

1. If F is $\emptyset$, then there are no solutions, never mind optimal solutions. Constraints contradict.
2. When F is unbounded, there may be no finite minimum. For example minimize x_1 subject to $x_1 \leq 1$. However, if we had to minimize x_1 subject to $x_1 \geq 1$, then F is still unbounded, but there is a finite solution.
3. There is an optimal solution x and we say $v^* = c^T x$.

4.2 *Algorithm

This is really more of a mathematical procedure than an algorithm. We can take a $v \in \mathbb{R}$ which is a guess that hopefully doesn't over estimate the optimal value v^*. Now we have $c^T x = v$, which by virtue of fixing v constrains x to lie somewhere in a hyperplane we'll call H_v. Now, if $v < v^*$, $H_v \not\cap F$, otherwise v^* would not be the optimum. So then we may progress by guessing $v + \epsilon$, effectively moving H_v up a little bit. After some iterations, we will see $H_v \cap F$, which would put v within ϵ of v^*, and we are done.

4.3 Remarks

To think a bit more about this geometrically, c is normal to H_v and governs its orientation. In the two dimensional figure above H_v would be a line with slope perpendicular to c; as the line moves up in the x_1 x_2 plane it will eventually intersect the yellow region F. c drives x in some particular direction for which F only permits us to go so far.

In general the intersection with F will occur at a vertex of the polytope, which will become useful because this vertex is the solution to some system of equations. This motivates our interest in solving systems of equations, which we now turn to.

5 Solving Systems of Equations

We're interested in solving $Ax = b$ where A is an m by n matrix. The core algorithm we have for doing this is Gaussian Elimination.

5.1 Example

$$3x_1 + 7x_2 + 2x_3 \;...\; + 4x_n = 9 \tag{1}$$
$$2x_1 + 4x_2 + 6x_3 \;...\; + 2x_n = 1 \tag{2}$$
$$... \tag{more equations}$$
$$x_1 + 8x_2 + 3x_3 \;...\; + 5x_n = 0 \tag{n}$$

Here we could solve the first equation for x_1, and then plug into the second equation to eliminate x_1 from it. Then we could solve the second equation for x_2 and so on[3] until we get to the n^{th} equation, which we could then solve for x_n and plug back in to get the rest.

5.2 Remarks

First we wish to understand the complexity of the solution. Suppose we even start with all integers defining the system, how many bits do we need to represent x? In particular, we'd like to prove that the number of bits is not exponential in n because if it is any algorithm to solve the system would also be exponential in runtime simply because of the description complexity of the output. We'd also like to better understand the cases when Gaussian Elimination fails.

[3] If all goes well at least.

6 Looking Ahead

Here we will just establish a list of facts from linear algebra that we'll be using. If any are a surprise, they would be worth reviewing.

If A is a square matrix, the following are equivalent:

1. A is invertible
2. $\text{Det}(A) \neq 0$
3. Columns of A are linearly independent
4. Rows of A are linearly independent
5. $Ax = b$ has a unique solution

Next time we'll continue by proving the following:
Claim 4. *The solution to $Ax = b$ has polynomial length description in n if A is composed of integers.*

We'll also start talking about duality and algorithms for linear programming.