Lecture 5

Ishaan Lal

September 27, 2024

1 Introduction

We'll now step away from the SAT problem for a bit. But don't fret! We will tie it back in later! Today, we will start looking at "high-level" solvers.

Working with SAT was great due to its universality, but it truly felt very low level – having to specify variables and clauses. Instead, we will now choose to specify constraints as a problem in something closer to mathematical language.

2 Linear Programming

Consider the following problem that we will return to throughout the section:

Problem

You are going to be attending a potluck. You want to take a meal with ≥ 5000 calories and ≤ 200 mg of sodium. Additionally, you want to spend as little money as possible.
You can bring any non-negative **real-value** (not necessarily integer) amount of the following items, so long as they abide to the above constraints:

Item	Price / kg	Calories / kg	Sodium / kg
Rice	1.25	750	15
Pasta	1.65	1200	35
Couscous	1.35	1000	60

For example, I could choose to bring 0kg of Rice, 4kg of Pasta, and 0.75kg of Couscous, which would yield $0 \cdot 750 + 4 \cdot 1200 + 0.75 \cdot 1000 = 5550$ calories and $0 \cdot 15 + 4 \cdot 35 + 0.75 \cdot 60 = 185$ kg of sodium. This solution abides to the constraints posed, and its total price is $0 \cdot 1.25 + 4 \cdot 1.65 + 0.75 \cdot 1.35 \approx 7.61$

For a bit of vocabulary, a **linear function** is a function of the form:

$$f(x_1, ..., x_n) = a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + a_n x_n$$

This can be thought of as an equation with only the variables, coefficients, and a constant. In other words, there are no "powers" on any of the variables. For the linear algebra inclined folks, a linear function can be expressed as the product of a row-vector of coefficients/constants with a column vector of variables.

$$f(x_1, ..., x_n) = A \cdot X \qquad \text{with} \quad A = \begin{bmatrix} a_0 & a_1 & ... & a_n \end{bmatrix}, \qquad X = \begin{bmatrix} 1 \\ x_1 \\ ... \\ x_n \end{bmatrix}$$

A **linear inequality** has the form:

$$a_1x_1 + a_2x_2 + \ldots a_nx_n \geq b \quad \text{or} \quad a_1x_1 + a_2x_2 + \ldots a_nx_n \leq b$$

A **linear program (LP)** is a special class of optimization problems with:

- The **GOAL** being to optimize a linear function. We will either *minimize* or *maximize* a function
- Existence of constraints given as **linear inequalities**

Example

As an example, an instance of a linear program may look like:

$$\begin{aligned} \text{minimize} \quad & 3x_1 + 2x_2 - 4x_3 + 5 \\ \text{subject to} \quad & x_1 \geq 2 \\ & x_2 + 2x_3 \leq 10 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Solution: We can actually argue a solution by hand. First, note that we must have $x_1 = 2$, because as x_1 increases, so does our *objective function* (the function we want to minimize). Additionally, we can note that the larger x_3 is, the smaller our objective function becomes. And we want x_2 to be as small as possible. This can be obtained with $x_2 = 0$ and $x_3 = 5$.
It turns out that this is the optimal solution.

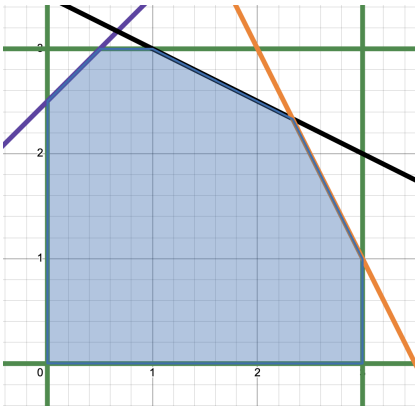
However, not all functions are easy to work with. The more variables we have, the more complex the problem becomes from a human perspective. Same goes for constraints.

Consider the following linear program:

$$\begin{aligned} \text{maximize} \quad & 2x_1 + 5x_2 \\ \text{subject to} \quad & 0 \leq x_1, x_2 \leq 3 \\ & -2x_1 + 2x_2 \leq 5 \\ & x_1 + 2x_2 \leq 7 \\ & 2x_1 + x_2 \leq 7 \end{aligned}$$

This is a challenging problem to approach by hand. But a good first step may be to find the "feasible region" – that is, the region of (x_1, x_2) values that satisfy all of the constraints. Once we have this region, we can then find the point within the region that maximizes the function.
Since our function only has two variables, consider the Cartesian plane, where the x-axis represents the

value of x_1 and the y-axis represents the value of x_2. Then, we can plot our constraints as lines:



Here, the blue area represents the *feasible region*. However, how do we know which point in the interior maximizes our function?

Brilliant Observation

A Brilliant observation is that the maximum **MUST** occur at one of the corners of the region!

What? How? The mathematical proof can be found, but we will sit content with a verbal, hand-wavy proof.
Instead of viewing our feasible region as just a set of points for us to test out, think of the feasible region as a mountain – where each point within the region has its own *elevation*, where the elevation is given by the functions value. That is,

$$\text{elevation} = 2x_1 + 5x_2$$

Then, we want to find the point with highest elevation. Critically, our function is **linear**, meaning that in the context of our mountain, it is just a simple slope. Thus, consider an arrow pointing in the direction of the slope, and from an interior point within the region, trek the mountain as far as you can in the direction of the arrow until you cannot anymore. If you are at a corner, then great. If you are at an edge, you must be able to traverse the edge to get to a corner that has elevation *at least as much* as the point you reached on the edge.

This observation gives way to some major results.

- First, it was found that **any LP** is polytime solvable.
- In practice, the algorithm used is called the Simplex algorithm, which is something that you can even do by hand! It is very reminiscent of matrix row reduction.
  - The Simplex algorithm was developed by George Dantzig in 1947.
  - In the worst case, it runs in exponential time
  - But for most instances/problems, it is practically fast.

The **Simplex Algorithm** essentially performs the algorithm we came up with. Plot out the feasible region, and then test out all of the corner points and choose the location that maximizes/minimizes the objective function.

NOTE: In our example, we were dealing with only 2 variables, which allowed for easy visualization. But as we increase the amount of variables, the visualization becomes finding a feasible region in hyperspace.

It wasn't until 1984 that a new algorithm, Karmarkar's algorithm, was developed that was polytime and also practically fast. It leveraged a theory called interior-point methods, which we will not go in to.

2.1 Back to The Potluck

Let's return to the potluck problem from before. We can formalize it as an instance of a linear program. In order to do this, we must identify three things:

1. The variables
2. The Objective Function (what we want to maximize/minimize)
3. The constraints.

To determine the variables we can ask ourselves: *What quantities have the ability to change?* When we answer this question, it tells us that the variables should be r, p, c, indicating the amount of rice, pasta, and couscous we purchase respectively.

Determining the objective function is straightforward: we want to minimize the price. The price, as a function of our variables is:

$$1.25 \cdot r + 1.65 \cdot p + 1.35 \cdot c$$

Lastly, we consider the constraints. First, we need ≥ 5000 calories, which can be expressed as:

$$750 \cdot r + 1200 \cdot p + 1000 \cdot c \geq 5000$$

and we must also have ≤ 200 mg of sodium, expressed as:

$$15 \cdot r + 35 \cdot p + 60 \cdot c \leq 200$$

Note that we also have an additional unspoken constraint of $r, p, c \geq 0$

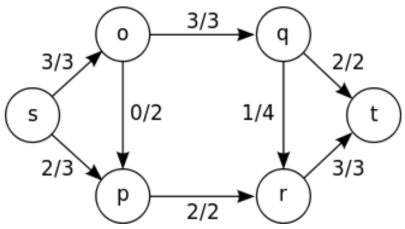
Thus, our formalized LP is:

$$\begin{aligned} \text{minimize} \quad & 1.25 \cdot r + 1.65 \cdot p + 1.35 \cdot c \\ \text{subject to} \quad & 750 \cdot r + 1200 \cdot p + 1000 \cdot c \geq 5000 \\ & 15 \cdot r + 35 \cdot p + 60 \cdot c \leq 200 \\ & r, p, c \geq 0 \end{aligned}$$

3 Max Flow

Let us now consider the Maximum Flow problem. This is a graph problem where, given a graph with edge capacities, source s and sink t, we wish to find the maximum amount of flow that can be sent from s to t so as to conserve flow (i.e. the flow coming into a vertex equals the flow going out of the vertex), and so as to not overflow an edge capacity. You can imagine this as starting at vertex s, and pouring water out from s along its outgoing edges, and any water that enters a node must leave the node. We want to figure out the most amount of water we can pour at the start. For more information, check out the first 3 minutes of this video.

The following graph shows a flow network.



Note: Solutions to the Flow problem are well known. For example, the Edmonds-Karp Algorithm solves the Max-flow problem and runs in time $\mathcal{O}(VE^2)$

3.1 Max Flow as LP

We choose to describe the Max Flow problem as an instance of LP.

First, we must choose our variables. We ask: *What quantities have the ability to change?* The answer to this question is: the amount of flow on each edge. Thus, our variables are:

$$f_{uv} = \text{total flow along edge } (u, v)$$

Our objective is to maximize the amount of flow we send out from our source s. Let $N_{\text{OUT}}(s)$ denote the neighbor set of s. That is, $N_{\text{OUT}}(s) = \{x \mid (s \to x) \in E\}$ contains all vertices adjacent to s. Then, our objective is:

$$\text{maximize} \quad \sum_{v \in N_{\text{OUT}}(s)} f_{sv}$$

Next, we turn to our constraints. The first constraint is that the amount of flow on each edge is non-negative and does not exceed its given capacity. Let $c(u, v)$ denote the capacity on edge (u, v). Then, we have:

$$0 \leq f_{uv} \leq c(u, v) \quad \forall (u, v) \in E$$

The conservation constraint is a bit more difficult to work with. We want to say for a non-s, t vertex u, the amount of flow that *enters* u is equal to the amount of flow that *exits* u. Let $N_{\text{IN}}(u)$ denote the set of vertices such that $v \in N_{\text{IN}}(u)$ if $v \to u \in E$. Then, we have:

$$\sum_{v \in N_{\text{IN}}(u)} f_{vu} = \sum_{v \in N_{\text{OUT}}(u)} f_{uv} \quad \forall u \in V - \{s, t\}$$

Or equivalently:

$$\sum_{v \in N_{\text{IN}}(u)} f_{vu} - \sum_{v \in N_{\text{OUT}}(u)} f_{uv} = 0 \quad \forall u \in V - \{s, t\}$$

And with this, we have a fully formed LP that we can go ahead and solve.

4 Integer (Linear) Programming

In the world of linear programming, one massive assumption we made was that fractional values are acceptable. And this is okay.

Integer Linear Programs (ILPs) are linear programs with the *additional constraint* that variables must take integer values.

Visually, recall our "feasible region" from before. Whereas before, any point within our region could be a potential solution, in ILP, the only potential solutions are points within the feasible region **that also have integer coordinates**.

Why would we ever care for an ILP when we have LP? Well, in the real world, items often come in discrete units. From our potluck example, suppose the information was given with respect to "bags" of the item instead of kilograms. It doesn't make much sense to buy 3.2 bags of rice.

ILP Example

Suppose we have 45 dollars to buy pies (5 dollars each) and cakes (9 dollars each). We can resell pies for 5 dollars of profit, and cakes for 8 dollars of profit. But we can only buy at most 6 items. How can we maximize profit?

We can write this as an instance of an ILP:

$$\begin{aligned} \text{maximize} \quad & 5p + 8c \\ \text{subject to} \quad & 5p + 9c \leq 45 \\ & p + c \leq 6 \\ & p, c \geq 0 \\ & p, c \in \mathbb{Z} \end{aligned}$$

4.1 Solving ILPs?

How do we solve ILPs? One naive idea is to use our solver for LP! After all, ILP and LP are very similar problems, and we already have a solution to LP.

Suppose we do this, and our solution returns the optimal values for our variables, but some of the variables are fractional when they need to be integers. What do we do? We may choose to round these values to the nearest integers that remain within the feasible region. This doesn't work.

- Bad News 1: We can construct ILPs whose rounded LP solution is *arbitrarily* far away from the optimal.
- Bad News 2: ILP is NP-complete!

5 Mixed Integer Program

Often, we will generalize ILP to **MIP (Mixed Integer Programming)**. With MIP, some variables may be constrained to integers, and some may not. Still, all objective functions and constraints are linear. We'll just talk about MIP since it encapsulates ILP.

5.1 Capital Budgeting Problem

To warm up our skills for MIP, let's consider the following problem:

- There are n possible investments, each with some associated utility/value v_i
- There are m resources, each with amount a_j
- Investment i costs c_{ij} units of resource j
- We want to make investments in a way so as to **maximize value**.

As we did with LP problems, we first need to choose our variables. Again, we ask ourselves: *What has the ability to change?*

The answer to this question may not be immediate, but it is: whether or not we choose each of the investments. Formally, our variables are x_i where:

$$x_i = \begin{cases} 1 & \text{we pick the } i^{th} \text{ investment} \\ 0 & \text{else} \end{cases} \qquad \forall i \in [1..n]$$

This idea of using "indicator" variables is **very** common and very useful in MIP and modeling in general.

Our objective function is to maximize value. This is simply the sum of all investments chosen:

$$\text{maximize} \quad \sum_i v_i x_i$$

Note that the investments that we do not choose do not contribute to the sum, because they would have $x_i = 0$

Lastly, we consider the only constraint: for each resource, the amount of it that we use does not exceed the amount available:

$$\sum_i c_{ij} \cdot x_i \leq a_j \quad \text{for each resource } j$$

5.2 Some Additional Constraints

Suppose we added an additional constraint: that we *need* to invest in investment i in order for us to be able to invest in investment j. How can we encode this as a constraint?

Let's consider if $x_i = 0$ – we do not invest in i. Then, we MUST have $x_j = 0$.
However, if $x_i = 1$, then we can have $x_j = 0$ or we could have $x_j = 1$.
The simplest way of expressing this situation is:

$$x_i \geq x_j$$

Suppose we had a constraint that investments i, j, k are conflicting. That is, we can only have at most one of these three investments. Then naturally, we would add the constraint:

$$x_i + x_j + x_k \leq 1$$

5.3 CPU Job Assignment Problem

Lastly, we consider the CPU Job Assignment Problem described as:

- There are n jobs that must be completed
- There are m CPUs available to do the jobs.
- Each CPU can do at most one job.
- There is a cost associated with running a particular job on a particular CPU
- How do we assign jobs to CPUs to minimize the total cost?

We start with defining our variables:

$$x_{c,j} = \begin{cases} 1 & \text{if CPU } c \text{ gets job } j \\ 0 & \text{else} \end{cases}$$

The natural constraint to impose is that each CPU gets at most 1 job, formalized as:

$$\text{For each CPU } c, \sum_{j \in \text{jobs} x_{c,j}} \leq 1$$

There is, however, one more constraint. In the current formulation, a job *could* be assigned to multiple CPUs. Thus, we add the constraint that each job gets exactly one CPU:

$$\text{For a job } j^\star, \text{ over all CPUs, exactly one of } x_{c_1,j^\star}, x_{c_2,j^\star}, ..., x_{c_n,j^\star} \text{ equals 1}$$