CS 3510 Algorithms 4/11/2024

Lecture 18: Linear Programming

Lecturer: Abrahim Ladha *Scribe(s): Saigautam Bonam*

Linear programming is a class of problems in which you have a linear set of constraints, multiple variables, and an objective function. The goal is to find a solution to the variables that maximizes the objective function while satisfying all constraints.

Suppose you sell items. Let's say item 1 sells for 1 dollar, item 2 for 6 dollars (item 2 is a luxury item). Your factory can currently produce only 400 items a day. The factory also cannot make more than 200 of item 1 and 300 of item 2 due to sanctions. We may present this problem as the following linear program:

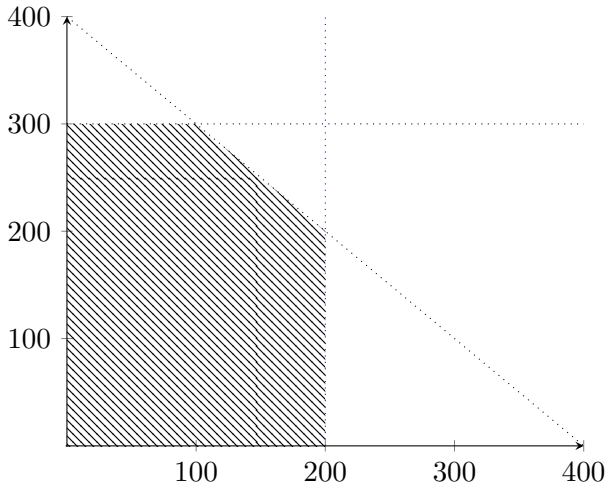
Objective Function: $\max\{x_1 + 6x_2\}$, subject to the following constraints:

$$\begin{aligned} x_1 &\leq 200 \\ x_2 &\leq 300 \\ x_1 + x_2 &\leq 400 \\ x_1, x_2 &\geq 0 \end{aligned}$$

We can also represent these attributes as the following matrices:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \; b = \begin{bmatrix} 200 \\ 300 \\ 400 \end{bmatrix} \; c = \begin{bmatrix} 1 \\ 6 \end{bmatrix}$$

where you want to maximize $c^T x$ subject to $Ax \leq b$ and $x \geq 0$. A represents the LHS of the constraints, and is a $m \times n$ matrix where m is the number of constraints, and n is the number of variables. b is a $m \times 1$ vector represents the RHS of the constraints, while c is a $n \times 1$ vector represents the coefficients of the objective function. We can plot the constraint inequalities and find the region in which all are satisfied below.



This is called the **feasible region**. The optimal solution would be when the line corresponding to $x_1 + 6x_2$ intersects the shaded region with the highest value. That would be at $(x_1, x_2) = (100, 300)$. This makes sense intuitively; since the coefficient is higher for item 2, we would want to satisfy the 400-item constraint by decreasing quantity of item 1 instead of item 2.

The optimal solution of a linear program (LP) is only unachievable when the constraints are either infeasible or unbounded. For an infeasible example, the constraints $x_1 \leq -1$ and $x_1 \geq 1$ will not be able to be satisfied. For an unbounded example, consider the only constraint is $x_1, x_2 \geq 0$ and the objective function as $\max\{x_1 + x_2\}$. There is no bounded solution.

We can write the standard form for an LP as follows using the A, b, c matrices defined earlier:

1. We wish to find solution $x = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$
2. The objective function is $\max c^T x$.
3. The objective function is subject to constraints $Ax \leq b$ and $x \geq 0$.

1 Three Classic Examples

Surprisingly, many problems can be represented by LPs.
Shortest Path: Suppose in a graph G we want to compute the shortest $s-t$ path. Let d_v be the shortest path from s to v. Note that $d_s = 0$. We may add constraints $d_v \leq d_u + w(u, v)$ for pairs of vertices u, v in G that share an edge. We can write the LP as follows:

$$\begin{aligned} \text{Objective Function} : \max d_t \\ \text{subject to} \quad d_v \leq d_u + w(u,v) \quad \forall (u,v) \in E \\ \text{and } d_s = 0 \end{aligned}$$

Why do we maximize instead of minimize in the objective function? $d_t = 0$ is the minimum solution, but the rest of the constraints have $d_t = \min_{u,(u,t)\in E}\{d_u + w(u,t)\}$ need to enforce that d_t is actually a path! d_t is the largest value less than or equal to all the paths set.

Max Flow: Given a flow network G where edges have capacities $c(u, v) \geq 0$, we want to maximize the flow from s to t. Here's the LP:

$$\text{Obj. Fn (sum of flows from s)} : \max\{\sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)\}$$

$$\text{subj. to constraints for all edges } f(u,v) \leq c(u,v)$$

$$f(u,v) \geq 0$$

$$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v) \quad \text{(flow conservation)}$$

The second term of the objective function is usually not required, as the incoming flow to s is usually 0 in most graphs we've seen, as s is a source. However, it's still necessary to write it in case it's not a well-formed graph. The constraints regarding the flow $f(u,v)$ ensure that the max flow through an edge is the capacity of the edge, and that there is nonnegative flow. Note that a constraint like $x = y$ is not in standard form, but can be transformed into one with constraints $x - y \leq 0$ and $y - x \leq 0$

Knapsack. We're given items (v_i, w_i) where we want to maximize the value of the item while ensuring the weight is below some constraint. Knapsack is already formatted as an LP! Consider the variables $x_1 \ldots x_i$ denoting a 1 if we choose an item and 0 and if we don't.

$$\text{Obj. Fn } : \max(\sum v_i x_i)$$

$$\text{subj. to constraints } \sum w_i x_i < W$$

$$\text{and } 0 \leq x_i \leq 1$$

Note this is technically not an LP, as you cannot take a fractional amount of items. It is an ILP.

2 Simplex

How do we solve LPs? Let's go over an algorithm to solve them called **Simplex**. Think of it as a traversal over the polytope of the feasible region. It is necessary that one of the corners of our polytope must be the optimal solution for our objective function. This is because our objective function is linear. Think about it like this: if you're on an edge and looking in two directions, there's one direction that will lead to a greater result and another that will lead to a smaller result. Choose the direction that leads to a larger result, and that will stop only when the edge stops. Here's the Simplex algorithm that revolves around the same idea; choose the direction that maximizes the objective function:

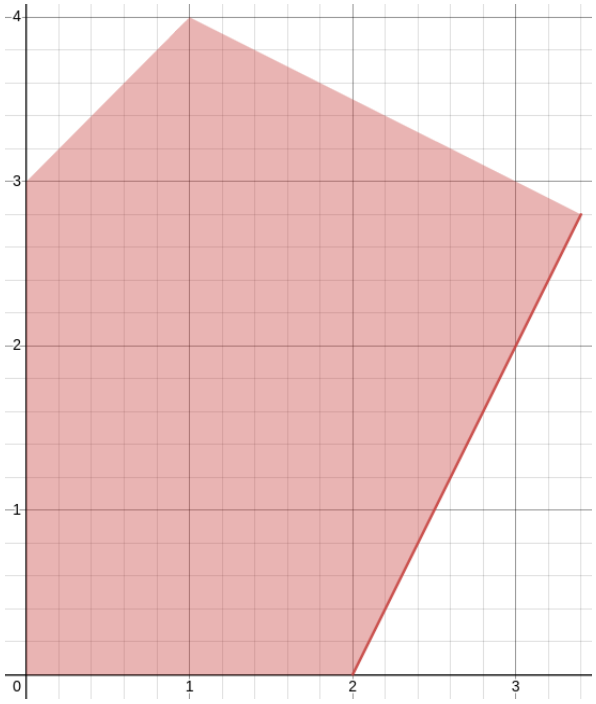
```
begin corner v = (0, 0)
while True:
    if any corner v' neighbor of v is more optimal:
        v = v'
    else
        break
```

To more easily determine v', we can change the origin to the current vertex v repeatedly by using a modified coordinate system. Let's look at this algorithm in action. Consider the following LP:

$$\text{Obj. Fn} : \max\{2x_1 + 5x_2\}$$

$$\begin{aligned} \text{subj. to constraints } & 2x_1 - x_2 \leq 4 && \text{(a)} \\ & x_1 + 2x_2 \leq 9 && \text{(b)} \\ & -x_1 + x_2 \leq 3 && \text{(c)} \\ & x_1 \geq 0 && \text{(d)} \\ & x_2 \geq 0 && \text{(e)} \end{aligned}$$

Finding the intersection of all these constraints gives us this polytope:



Let's start with the corner $(0, 0)$. Looking at the two adjacent corners, these are $(0, 3)$ and $(2, 0)$. Plugging these into our objective function, we can see that $(0, 3)$ gives us a higher value, which makes sense given the coefficient for x_2 is larger than the coefficient for x_1. We move to vertex (0, 3). From there, we can see the next corner is (1, 4), which results in a larger value. We move to (1, 4). However, from (1, 4), we can see that the next coordinate, which is (3.4, 2.8), actually results in a smaller value in our objective function. As a result, we stop at (1, 4), which is our solution to the LP.

This variant of LP allows solutions to be real numbers, which guarantees by linearity that the solutions can be found by moving through the edges of the feasible region. This makes

LP, on average, a polynomial time problem to solve. However, a variant of LP is **Integer Linear Programming** (ILP), which corresponds to Knapsack!

ILP is an NP-complete decision variant of LP where the solutions must be integers. Let's prove it's NP-Hard. We can reduce from Vertex Cover instead of Knapsack. Recall the definition of Vertex Cover: given input G, g, it returns if there exists a set $S \subset V$ of size g such that every edge has an endpoint in S. For the reduction: for each edge (u, v) in G, add variables x_u, x_v which can take on values 0 or 1 and the constraint $x_u + x_v \geq 1$, which symbolizes choosing one vertex. The objective function is to maximize the sum of the variables! If we can get at least g as a result from the ILP, we return True for Vertex Cover. Otherwise, we return False.