DS 320 Algorithms for Data Science **Lecture #18 Worksheet**
Spring 2022 Prof. Kira Goldner

Linear Programming I

Why Linear Programming rocks:

- Incredibly general: Every problem we've seen so far can be formulated as a linear program.
- Computational tractable
  - In theory: Can be solved in polynomial time
  - In practice: Fast with input sizes up into the millions!
- Contains many properties that can be turned into useful algorithmic paradigms and analysis:
  - Duality:
    - Solve an easier equivalent problem.
    - How do we know when we're done?
  - Complementary Slackness and Strong Duality: something is optimal!

How to Think About Linear Programming

Comparison to Systems of Linear Equations

Think back to linear systems of equations. Such a system consists of m linear equations in real-valued variables $x_1, \ldots, x_n$:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{12}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{22}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{m2}x_n &= b_m. \end{aligned}$$

The a_{ij}'s and the b_i's are given; the goal is to check whether or not there are values for the x_j's such that all m constraints are satisfied. We used Gaussian elimination; "solved" meant that the algorithm returns a feasible solution, or correctly reports that no feasible solution exists.

What about inequalities? The point of linear programming is to solve systems of linear equations *and inequalities*. Moreover, when there are multiple feasible solutions, we would like to compute the "best" one.

Ingredients of a Linear Program

There is a convenient and flexible language for specifying linear programs, and we'll get lots of practice using it during this lecture. Sometimes it's easy to translate a computational problem into this language, sometimes it takes some tricks (we'll see examples of both).

To specify a linear program, you need to declare what's allowed and what you want.

Ingredients of a Linear Program

a. *Decision variables* $x_1, \ldots, x_n \in \mathbb{R}$.

b. *Linear constraints*, each of the form

$$\sum_{j=1}^{n} a_j x_j \quad (*) \quad b_i,$$

where $(*)$ could be $\leq, \geq$, or $=$.

c. A *linear objective function* of the form

$$\max \sum_{j=1}^{n} c_j x_j$$

or

$$\min \sum_{j=1}^{n} c_j x_j.$$

Comments:

- The a_{ij}'s, b_i's, and c_j's are *constants*, part of the input, hard-wired into the linear program (like 5, -1, 10, etc.).
- The x_j's are free, and it is the job of a linear programming algorithm to figure out the best values for them.
- When specifying constraints, there is no need to make use of both "$\leq$" and "$\geq$"inequalities—one can be transformed into the other just by multiplying all the coefficients by -1 (the a_{ij}'s and b_i's are allowed to be positive or negative).
- Equality constraints are superfluous, in that the constraint that a quantity equals b_i is equivalent to the pair of inequality constraints stating that the quantity is both at least bi and at most b_i.
- There is also no difference between the "min" and "max" cases for the objective function—one is easily converted into the other just by multiplying all the c_j's by -1 (the c_j's are allowed to be positive or negative).

What's not allowed in a linear program? Terms like $x_j^2, x_j x_k, \log(1 + x_j)$, etc. So whenever a decision variable appears in an expression, it is alone, possibly multiplied by a constant (and then summed with other such terms). While these linearity requirements may seem restrictive, we'll see that many real-world problems can be formulated as or well approximated by linear programs.

A Simple Example

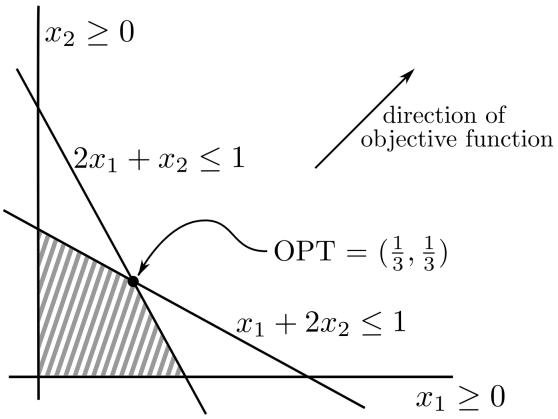


Figure 1: A toy example of a linear program.

To make linear programs more concrete and develop your geometric intuition about them, let's look at a toy example. (Many "real" examples of linear programs are coming shortly.) Suppose there are two decision variables x_1 and x_2—so we can visualize solutions as points (x_1, x_2) in the plane. See Figure 1. Let's consider the (linear) objective function of maximizing the sum of the decision variables:

$$\max x_1 + x_2.$$

We'll look at four (linear) constraints:

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \\ 2x_1 + x_2 &\leq 1 \\ x_1 + 2x_2 &\leq 1. \end{aligned}$$

The first two inequalities restrict feasible solutions to the non-negative quadrant of the plane. The second two inequalities further restrict feasible solutions to lie in the shaded region depicted in Figure 1. Geometrically, the objective function asks for the feasible point furthest in the direction of the coefficient vector $(1, 1)$—the "most northeastern" feasible point. Eyeballing the feasible region, this point is $(\frac{1}{3}, \frac{1}{3})$, for an optimal objective function value of $\frac{2}{3}$. This is the "last point of intersection" between a level set of the objective function and the feasible region (as one sweeps from southwest to northeast).

Geometric Intuition

While it's always dangerous to extrapolate from two or three dimensions to an arbitrary number, the geometric intuition above remains valid for general linear programs, with an arbitrary number of dimensions (i.e., decision variables) and constraints. Even though we can't draw pictures when there are many dimensions, the relevant algebra carries over without any difficulties. Specifically:

a. A linear constraint in n dimensions corresponds to a halfspace in $\mathbb{R}^n$. Thus a feasible region is an intersection of halfspaces, the higher-dimensional analog of a polygon.[1]

b. When there is a unique optimal solution, it is a vertex (i.e., "corner") of the feasible region.

A few edge cases:

a. There might be no feasible solutions at all. For example, if we add the constraint $x_1 + x_2 \geq 1$ to our toy example, then there are no longer any feasible solutions. Linear programming algorithms correctly detect when this case occurs.

b. The optimal objective function value is unbounded ($+\infty$ for a maximization problem, $-\infty$ for a minimization problem). Note a necessary but not sufficient condition for this case is that the feasible region is unbounded. For example, if we dropped the constraints $2x_1 + x_2 \leq 1$ and $x_1 + 2x_2 \leq 1$ from our toy example, then it would have unbounded objective function value. Again, linear programming algorithms correctly detect when this case occurs.

c. The optimal solution need not be unique, as a "side" of the feasible region might be parallel to the levels sets of the objective function. Whenever the feasible region is bounded, however, there always exists an optimal solution that is a vertex of the feasible region.[2]

A Case Study: Vertex Cover

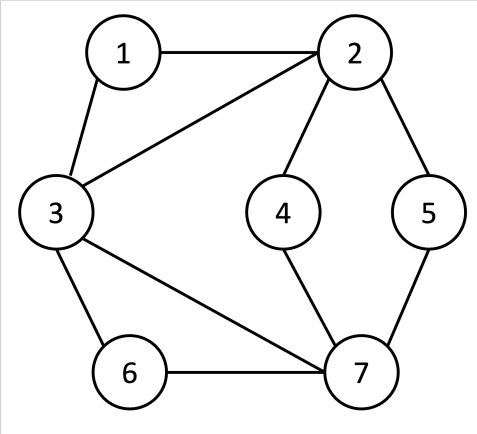
The Vertex Cover Problem

Given a graph $G = (V, E)$, we say that a set of nodes $S \subseteq V$ is a *vertex cover* if every edge $e = (i, j) \in E$ has at least one endpoint i or j in S. Our goal is to find a *minimum* vertex cover.

Given a graph G and a number k, does G contain a vertex cover of size at most k?

[1] A finite intersection of halfspaces is also called a "polyhedron;" in the common special case where the feasible region is bounded, it is called a "polytope."

[2] There are some annoying edge cases for unbounded feasible regions, for example the linear program $\max(x_1 + x_2)$ subject to $x_1 + x_2 = 1$.



In this graph, the *minimum* vertex cover is

This is the same graph from last time when we discussed Independent Set. Do we notice any relationship?

Claim 1.

Corollary 1.

Corollary 2.

Vertex Cover as an Integer Program

a. *Decision variables:* What are we try to solve for?

b. *Constraints:*

c. *Objective function:*

Vertex Cover as a Linear Program

Claim 2. Let S^* denote the optimal vertex cover of minimum weight, and let x^* denote the optimal solution to the Linear Program. Then $w^T x^* \leq w(S^*)$.

Claim 3. The set $S = \{i : x_i \geq 0.5\}$ is a vertex cover, and $w(S) \leq 2\, w^T x^*$.

Writing Problems We Know as Linear Programs

Independent Set

Given a graph $G = (V, E)$, each vertex i has weight w_i, find a maximum weighted *independent set*. S is an independent set if it does not contain both i and j for $(i, j) \in E$.

Knapsack

Given n items, each item i with value v_i and weight w_i, select a set S that contains maximum value but has total weight of at most W.