20 Linear Programming

In this chapter we will introduce a powerful algorithmic technique called linear programming which provides a general framework that can solve a wide variety of problems, including ones we have previously encountered like shortest paths and maximum flow. In this chapter we will focus on applications of linear programming, treating algorithms for linear programming as black boxes. In the next chapter, we will look in more detail at techniques for solving linear programs.

20.1 What is a Linear Program?

Contrary to our modern concept of "programming", a linear program does not involve any code. However, when the term was coined by Tjalling Koopmans in the 1940s, computers were in their infancy, and "programming" meant something more like "optimization."

A linear program P consists of:

- A list of n variables $x_1, \ldots, x_n$,
- A list of linear constraints in those variables, where each constraint is either a linear $\leq$ inequality, a linear $\geq$ inequality, or a linear equation (strict inequalities (e.g. $<$) are not allowed), and
- An objective which is either minimizing or maximizing a linear function.

A linear function is one of the form $a_1x_1 + a_2x_2 + \cdots + a_nx_n$, where $a_1, \ldots, a_n \in \mathbb{R}$ are constants (which could be positive, negative, or zero). That is, a linear function is the sum of terms each of which is a variable multiplied with a constant. We can view a linear function as a function f operating on the vector $x = x_1, \ldots, x_n$, so that our constraints have the form $f_i(x) \leq b_i$, $f_i(x) \geq b_i$, or $f_i(x) = b_i$, where $f_1, \ldots, f_m$ are linear functions and $b_1, \ldots, b_m \in \mathbb{R}$ are constants. Note that a linear function can include terms like $x_1 - x_2$ (this is the same as $x_1 + (-1)x_2$), but cannot include expressions like x_1^2, $x_1 \cdot x_2$, or x_1/x_2.

For example, this is a linear program:

Figure 17: A linear program

$$\begin{array}{lrcl} \text{minimize} & x_1 - 2x_2 + x_3 & & \\ \text{subject to} & 2x_1 + 5x_3 + x_4 & \leq & 8 \\ & 3x_1 - 4x_2 + 2x_4 & \geq & -3 \end{array}$$

Here the variables are x_1, x_2, x_3, x_4, the constraints are $2x_1+5x_3+x_4 \leq 8$ and $3x_1-4x_2+2x_4 \geq -3$, and the objective is to minimize $f(x) = x_1 - 2x_2 + x_3$ (we call the function f we are trying to maximize/minimize the objective function).

We call an assignment of values from $\mathbb{R}$ to the variables of an LP a **feasible solution** to the LP if it satisfies all of the constraints, regardless of what objective value it achieves. For example, $x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0$ is a feasible solution to the LP in Figure 17.

Figure 18: An infeasible linear program

$$\begin{array}{lrcl} \text{minimize} & x_1 + x_2 & & \\ \text{subject to} & x_1 & \leq & 5 \\ & 2x_1 + x_2 & \geq & 10 \\ & x_1 + x_2 & = & -3 \end{array}$$

Some LPs have no feasible solutions. For example, consider the LP in Figure 18. Combining the third constraint with the second we have that

$$2x_1 + x_2 = x_1 + (x_1 + x_2) = x_1 - 3 \geq 10.$$

$x_1 - 3 \geq 10$ implies that $x_1 \geq 13$. But by the first constraint, we have that $x_1 \leq 5$. There is no value we can pick for x_1 which is simultaneously greater than or equal to 13 and less than or equal to 5, so this LP has no feasible solutions. We call a linear program with no feasible solutions **infeasible**, while a linear program with at least one feasible solution is called **feasible**. Note that whether or not an LP is feasible depends only on the constraints, not on the objective function.

We call a feasible solution an **optimal solution** if no other feasible solution has a better value for the objective function ("better" meaning "larger" in a maximization LP and "smaller" in a minimization LP). We call the objective value of an optimal solution the **optimal value** of the LP. For example, $x_1 = 3, x_2 = 4$ is an optimal solution for the LP below. Note that an LP may have more than one optimal solution. This LP has infinitely many, including $x_1 = 3.5, x_2 = 3.5$, $x_1 = 12, x_2 = -5$, and $x_1 = \pi$, $x_2 = 7 - \pi$. All of these optimal solutions attain the same objective value. The optimal value for this LP is 7.

$$\begin{array}{ll} \text{maximize} & x_1 + x_2 \\ \text{subject to} & x_1 + x_2 \leq 7 \end{array}$$

Note that all of the coefficients and constants being integers does not guarantee that a linear program has an integer optimal solution. For example, the unique optimal solution for the LP below is $x_1 = 16/5, x_2 = 3/5$, with an objective value of $19/5$.

$$\begin{array}{lll} \text{maximize} & x_1 + & x_2 \\ \text{subject to} & 2x_1 + & x_2 \leq 7 \\ & x_1 + & 3x_2 \leq 5 \end{array}$$

It is possible for a feasible linear program to have no optimal solution if it is possible to achieve arbitrarily large value for the objective value. For example, in the LP below, we can achieve an objective value of k, for any $k \in \mathbb{R}^+$ by setting $x_1 = -k$ and $x_2 = 2k$, so that our objective value is $x_1 + x_2 = k$, but $2x_1 + x_2 = 0 \leq 7$, so the constraint is satisfied. We call a linear program **unbounded** if there are feasible solutions with arbitrarily good objective values (where "good" means large for a maximization LP and "small" for a minimization LP), and **bounded** otherwise. Note that, unlike feasibility, whether or not a linear program is bounded depends on the objective function.

$$\begin{array}{ll} \text{maximize} & x_1 + x_2 \\ \text{subject to} & 2x_1 + x_2 \leq 7 \end{array}$$

We say that a linear program is in **standard form** (also sometimes called "canonical form") if:

- It has n inequalities of the form $x_i \geq 0$
- The remaining m inequalities are all of the form $a_{j1}x_1 + a_{j2}x_2 + \cdots + a_{jn}x_n \leq b_j$, where $a_{j1}, \ldots, a_{jn}$, and b_j are constants (i.e. the remaining inequalities are all $\leq$ inequalities with a constant on the right hand side).
- The objective is maximization (rather than minimization).

We can convert any LP into standard form via a few transformations:

1. If the objective is minimization, we can convert it into maximization by simply multiplying the objective function by -1. Minimizing $f(x)$ is the same as maximizing $-f(x)$.
2. To get all of our inequalities in the form $f_i(x) \leq b_i$, we first convert each equation $f(x) = b$ into two inequalities: $f(x) \leq b$ and $f(x) \geq b$. Then we convert each $\geq$ inequality into a $\leq$ inequality by multiplying both sides by -1 (since $f(x) \geq b \Rightarrow -f(x) \leq -b$).
3. To add the restriction that all of our variables are non-negative, we introduce two new variables y_i and z_i for each variable x_i in the original LP. We add the constraints $y_i \geq 0$ and $z_i \geq 0$, and we replace every occurrence of x_i in the constraints and objective function with $y_i - z_i$.

   To see why this works, we note that every real number can be expressed as the difference of two non-negative real numbers: to express a positive value k we can set $y_i = k$ and $z_i = 0$, while to express $-k$ we set $y_i = 0$ and $z_i = k$.

For example, suppose that we want to convert the LP below (which is the one from Figure 18) into standard form:

$$\begin{array}{ll} \text{minimize} & x_1 + x_2 \\ \text{subject to} & x_1 \leq 5 \\ & 2x_1 + x_2 \geq 10 \\ & x_1 + x_2 = -3 \end{array}$$

We begin by converting the objective into maximization:

$$\begin{array}{ll} \text{maximize} & -x_1 - x_2 \\ \text{subject to} & x_1 \leq 5 \\ & 2x_1 + x_2 \geq 10 \\ & x_1 + x_2 = -3 \end{array}$$

Next we convert the equation into two inequalities:

$$\begin{array}{ll} \text{maximize} & -x_1 - x_2 \\ \text{subject to} & x_1 \leq 5 \\ & 2x_1 + x_2 \geq 10 \\ & x_1 + x_2 \geq -3 \\ & x_1 + x_2 \leq -3 \end{array}$$

Then we convert the $\geq$ inequalities into $\leq$ inequalities:

$$\begin{array}{ll} \text{maximize} & -x_1 - x_2 \\ \text{subject to} & x_1 \leq 5 \\ & -2x_1 - x_2 \leq -10 \\ & -x_1 - x_2 \leq 3 \\ & x_1 + x_2 \leq -3 \end{array}$$

Finally, we replace x_1 and x_2 with the new variables y_1, y_2, z_1, z_2 to introduce non-negativity constraints to get a standard form version of the LP:

$$\begin{array}{ll} \text{maximize} & -y_1 + z_1 - y_2 + z_2 \\ \text{subject to} & y_1 - z_1 \leq 5 \\ & -2y_1 + 2z_1 - y_2 + z_2 \leq -10 \\ & -y_1 + z_1 - y_2 + z_2 \leq 3 \\ & y_1 - z_1 + y_2 - z_2 \leq -3 \\ & y_1 \ , \ z_1 \ , \ y_2 \ , \ z_2 \geq 0 \end{array}$$

An LP in standard form can be represented by:

- An $m \times n$ matrix A, where $A[i, j] = a_{ij}$ is the coefficient of x_j in the i'th constraint,
- A length m vector b, where b_i is the constant on the right hand side of the i'th inequality
- A length n vector c, where c_j is the coefficient of x_j in the objective function.

Under this representation, the LP can be compactly expressed as:

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{s.t.} & Ax \leq b \\ & x \geq 0. \end{array}$$

This formulation also allows us to analyze the LP using the tools of linear algebra. However, since linear algebra is not a prerequisite for this course, we will focus on solving and applying LPs over proving their properties. Nevertheless, we'll see some reasons to care about having LPs in standard form next chapter.

20.2 Formulating Problems as Linear Programs

So what are linear programs good for? Quite a lot, actually! Many of the problems we've seen in previous chapters can be seen as special cases of linear programming.

For example, recall the maximum flow problem from Chapter 18. A flow in a flow network is defined by the capacity and flow conservation constraints, both of which are based on linear functions of the flow through each edge. Thus, we can model max flow with the following LP, which has a variable x_e for each edge in G representing the amount of flow we sent through edge e (recall that $\delta_{in}(v)$ is the set of incoming edges to v, $\delta_{out}(v)$ is the set of outgoing edges of v, and c_e is the capacity of the edge e): Here our objective is to maximize the value of the flow, i.e. the net flow out of s.

Figure 19: A linear program for maximum flow

$$\begin{array}{lrll} \text{maximize} & \sum_{e\in\delta_{out}(s)} x_e - \sum_{e\in\delta_{in}(s)} x_e & & \\ \text{subject to} & x_e & \leq c_e & \forall\, e \in E \\ & x_e & \geq 0 & \forall\, e \in E \\ & \sum_{e\in\delta_in(v)} x_e - \sum_{e\in\delta_{out}(v)} x_e & = 0 & \forall\, v \in V \setminus \{s,t\} \end{array}$$

The single source shortest paths problem from Chapters 15 and 16 can also be modeled as a linear program. Again the key observation is that we already have a linear constraint on the shortest path distances. Remember that Dijkstra's algorithm and Bellman-Ford were both based on edge relaxations taking advantage of the fact that if $(u, v) \in E$ then $dist(s, v) \leq dist(s, u) + w(u, v)$, because we can always get to v by following the shortest $s \to u$ path and then following the edge (u, v). Thus, if we define a variable x_v to represent $dist(s, v)$ for each vertex v, then our variables should satisfy the constraint $x_v \leq x_u + w(u, v)$ (or equivalently $x_v - x_u \leq w(u, v)$) for each edge (u, v). Furthermore, we know that x_s should be 0.

It's not clear what our objective should be for the shortest paths LP. At first it seems like we should use a minimization LP, since we are looking for the shortest paths. However, our constraints impose no lower bounds on the distance values, so if our goal was minimization, then we could simply set x_v to 0 for every vertex, or even to negative values.

The constraints we have established guarantee that our distance values will not produce any tense edges, i.e. they cannot be too large. In particular, summing the constraints for the edges along a shortest $s \to v$ path $s = v_1, v_2, \ldots, v_{k-1}, v_k = v$ we have that

$$\sum_{i=1}^{k-1}(x_{v_{i+1}} - x_{v_i}) \leq \sum_{i=1}^{k-1} w(v_i, v_{i+1}) = dist(s, v).$$

Canceling out terms, the summation on the left side becomes $x_{v_k} - x_{v_1} = x_v - x_s = x_v - 0 = x_v$, so we have that $x_v \leq dist(s, v)$. Thus, our constraints guarantee that each variable x_v is less than or equal to the length of the shortest $s \to v$ path. To guarantee that we also have $x_v \geq dist(s, v)$, the right idea is to make our objective maximizing the sum of the variables x_v. This will ensure that, in an optimal solution for the LP, x_v will be exactly $dist(s, v)$ (assuming every vertex is reachable from s).

Putting these ideas together, we can write the following LP for computing the shortest paths from s.

Figure 20: A linear program for single-source shortest paths

$$\begin{array}{lrll} \text{maximize} & \sum_{v\in V} x_v & & \\ \text{subject to} & x_s & = 0 & \\ & x_v - x_u & \leq w(u,v) & \forall (u,v) \in E \end{array}$$

Note that this LP can handle negative weight edges. If there is a negative weight cycle, the LP will be infeasible (no assignment of distance values to the vertices in the cycle can satisfy the "no tense edges" constraints).

If there are vertices unreachable from s, then this shortest paths LP will be unbounded. To avoid this, before creating the LP we can run DFS (or BFS) from s to determine reach(s) and remove from G all of the vertices that s cannot reach.

Linear programming can also solve problems that we haven't seen before. One important example is a generalization of maximum flow known as *min-cost max-flow*. As in a regular max-flow problem, we are given a flow network consisting of a directed graph $G = (V, E)$ with special vertices s and t where each edge e has a capacity $c(e)$. Now, however, each edge e also has an associated cost or weight $w(e)$ (since we are already using $c(e)$ for capacity, we will stick to weights). Our goal is still to find a maximum flow from s to t (i.e. a flow with the largest capacity), however if there are multiple maximum flows, then we must output the one with the minimum total weight, where the weight of a flow f is defined as $\sum_{e \in E} w(e)f(e)$.

In order to model this problem as a linear program we need to figure out how we can encode our dual objectives of maximizing the flow capacity and minimizing the weight of the flow, when our LP can only have one objective function. The idea here is that since we are requiring the flow value to be the maximum possible, we can encode our requirement for the flow out of s as a *constraint* rather than the objective. In particular, we can first solve the LP from Figure 19 to find the max flow value OPT for G, then create the following LP for min-cost max-flow:

Figure 21: A linear program for min-cost maximum flow

$$
\begin{array}{lrll}
\text{minimize} & \sum_{e \in E} w(e) x_e & & \\
\text{subject to} & x_e \leq & c_e & \forall\, e \in E \\
& x_e \geq & 0 & \forall\, e \in E \\
& \sum_{e \in \delta_{in}(v)} x_e - \sum_{e \in \delta_{out}(v)} x_e = & 0 & \forall\, v \in V \setminus \{s, t\} \\
& \sum_{e \in \delta_{out}(s)} x_e - \sum_{e \in \delta_{in}(s)} x_e = & OPT &
\end{array}
$$

Like the linear program for regular max-flow, this LP will have an optimal solution which assigns each variable an integer value, assuming that all of the edge capacities are integers. We will not formally prove this fact, but you may assume it for the exercises and quizzes (you should *not* assume this for every LP you write, because it is not true for general LPs).

20.3 Algorithms for Linear Programming

In order to discuss the runtime for solving linear programs we first need to consider how to encode linear programs as strings (i.e. how do we define the input size for an algorithm that takes a linear program). Throughout this section we will assume that n is the number of variables in our LP and m is the number of constraints.

Similar to the assumptions we have previously made for array elements and edge weights in graphs, we will assume, unless otherwise specified, that when an algorithm takes a linear program as input, the values of the constants in the LP (e.g. the coefficients of the variables) will be integers whose values are polynomial in the number of variables (i.e. $O(n^c)$ for some constant c), so that they fit in a constant number of RAM model integer variables and we can perform arithmetic on them in constant time. It is not clear that this restriction guarantees that the optimal solution to our LP (if it exists) has a small representation, but it turns out that if all of the constants in the LP are polynomially-bounded integers, then if the LP is feasible and bounded, it has an optimal solution where all of the variables are assigned rational values which can be expressed as strings of length $O(n \log n)$.

The first algorithm for solving arbitrary linear programs was developed by George Dantzig in 1947 and is known as the simplex algorithm. We will discuss this algorithm in detail in the next chapter. While the theoretical guarantees on its runtime are not great (it can take exponential time

on carefully constructed bad inputs), the simplex algorithm is simple to implement and efficient in practice for many real-world LPs, so it is still commonly used.

The first polynomial time algorithm for solving a linear program is the ellipsoid method, developed in 1979 by Leonid Khachiyan. For a linear program with n variables , the ellipsoid method runs in time $O(n^7 m \log n)$. For almost any practical LP, this is far slower than the simplex method.

The fastest known algorithms for linear programming are based on "interior point methods" pioneered by Narendra Karmarkar in the 1980s. In particular, the current best-known algorithm was published by Cohen, Lee and Song in 2019. This algorithm solves a linear program using a series of matrix multiplications, and the runtime of the algorithm depends on the runtime for matrix multiplication. With the known results at the time that Cohen, Lee and Song published their algorithm, it ran in time $O((n+m)^{2.37})$. Very recent improvements in the runtime for matrix multiplication (in particular, an October 2026 paper by Amanov, van den Berg and Zuiddam) improves the runtime of Cohen, Lee, and Song's algorithm to $O((n+m)^{2.25+o(1)})$.

20.4 Comprehension Questions

1. Suppose that P is a linear program with a unique optimal solution. Which of the following must be true about P? (select all that apply)

   (a) P is feasible

   (b) P is bounded

   (c) P has the same number of variables as constraints

   (d) P is in standard form

2. Bureaucracy Inc. makes 3 products, A, B, and C, out of widgets, gizmos, and doodads. Each product requires different components and sells for a different price. The table below shows the quantities of widgets, gizmos, and doodads required to make each product, as well as the price each product sells for.

Product	Widgets	Gizmos	Doodads	Sell Price
Product A	2	3	0	10
Product B	4	0	2	15
Product C	3	5	1	25

   Bureaucracy Inc. currently has an inventory of 200 widgets, 100 gizmos, and 50 doodads, and would like to combine these to make products which sell for the maximum total amount. They want to formulate the optimal combination of products as a linear program with variables a, b, and c, representing the quantities of products A, B, and C. Which of the following makes the most sense as the objective function for this linear program?

   (a) Maximize $a + b + c$

   (b) Maximize $10a + 15b + 25c$

   (c) Maximize $200a + 100b + 50c$

   (d) Minimize $a + b + c$

   (e) Minimize $10a + 15b + 25c$

   (f) Minimize $200a + 100b + 50c$

3. Which of the following makes the most sense as the constraints for the linear program described in the previous problem?

(a)

$$\begin{aligned} 2a + 3b &\leq 10 \\ 4a + 2c &\leq 15 \\ 3a + 5b + c &\leq 25 \\ a, b, c &\geq 0 \end{aligned}$$

(b)

$$\begin{aligned} 2a + 3b &\leq 200 \\ 4a + 2c &\leq 100 \\ 3a + 5b + c &\leq 50 \\ a, b, c &\geq 0 \end{aligned}$$

(c)

$$\begin{aligned} 2a + 4b + 3c &\leq 10 \\ 3a + 5c &\leq 15 \\ 2b + c &\leq 25 \\ a, b, c &\geq 0 \end{aligned}$$

(d)

$$\begin{aligned} 2a + 4b + 3c &\leq 200 \\ 3a + 5c &\leq 100 \\ 2b + c &\leq 50 \\ a, b, c &\geq 0 \end{aligned}$$

4. Convert the following LP to standard form:

$$\begin{array}{ll} \text{minimize} & x_1 - 3x_2 \\ \text{subject to} & -2x_1 + 5x_2 = 7 \end{array}$$

5. Recall that in the linear program for single source shortest paths we defined a variable x_v for each vertex in the graph, designed to represent $dist(s, v)$. What was our objective function for this LP?

20.5 Exercises

1. The government of Algorithmica is attempting to pass a budget for the country. A budget consists of allocating a non-negative amount of money to each of x different programs, $p_1, \ldots, p_x$. Approving a budget requires unanimous support from the country's y senators, $s_1, \ldots, s_y$. Each senator s_i has a set of programs S_i they care about, and will only support the budget if those programs collectively receive at least \$10,000 in funding. The government has b dollars of tax revenue to allocate among all the programs. Give a polynomial time algorithm that takes as input the total available money, b, the list of programs $p_1, \ldots, p_x$ and the sets of senator preferences $S_1, \ldots, S_y$ and determines whether or not there exists a budget that will be approved by all of the senators.

2. In the min-weight perfect matching problem we are given a weighted bipartite graph $G = (L \cup R, E)$ and the goal is to find a perfect matching in G with minimum total weight (among all perfect matchings), or correctly report that no perfect matching exists. Note that the weight of a matching M is $\sum_{e \in M} w(e)$. Give a polynomial time algorithm to solve the min-weight perfect matching problem.

3. Suppose that you are given n points (x_i, y_i) in the Cartesian plane with integer coordinates (i.e. a scatter plot). Your goal is to fit a line through these points. A line is defined by an equation of the form $ax + b$. For given values of a and b we define the error of the line $ax + b$ at representing the given points as $\sum_{i=1}^{n} |ax_i + b - y_i|$. Give a polynomial time algorithm to find values for a and b which describe a line with minimum error for representing the given points.

4. You have been put in charge of shipping logistics for a shockingly successful new chain of stores selling identical tungsten cubes. You are in charge of making a plan to ship cubes from k different factories to ℓ different branches of your store. You are given the following:

   - An array of integers $F[1, \ldots, k]$ where $F[i]$ is the number of cubes produced at factory i each week
   - An array of integers $S[1, \ldots, \ell]$ where $S[i]$ is the maximum number of cubes that store i can sell in a week.
   - An array of integers $P[1, \ldots, \ell]$ where $P[i]$ is the price (in dollars) that store i sells each of its cubes for (some areas are willing to pay more than others)
   - A 2-D array of integers $C[1, \ldots, k][1, \ldots, \ell]$, where $C[i, j]$ is the cost (in dollars) to ship a single cube from factory i to store j (your business is extremely bad at taking advantage of economies of scale, so you ship each cube individually).

   Give a polynomial time algorithm to determine the maximum profit that you can earn in a week by shipping cubes to stores and selling them. Be careful to ensure that your solution ships an integer number of cubes to each store (nobody wants to buy half a cube).

5. Suppose that you are given a system of n linear inequalities of the form $\alpha_i x + \beta_i y \leq \gamma_i$. These inequalities define a convex polygon P in the 2-D plane.

   (a) Give an LP whose solution computes the maximum *perimeter* axis-aligned rectangle contained within P (if P is empty, your LP should be infeasible). Axis-aligned means that the sides of the rectangle must be horizontal and vertical.

   (b) Give an LP whose solution computes the maximum *area* axis-aligned *square* contained within P (if P is empty, your LP should be infeasible). Axis-aligned means that the sides of the square must be horizontal and vertical.