Chapter 26 Linear Programming

Brooklyn Bridge showing painters on suspenders. By Eugene de Salgnac, October 7, 1914. From NYC Municipal Archives. Public domain image.

Contents

Optimization problems are common in the real world, with many applications in business and science that involve maximizing or minimizing some goal subject to a set of given constraints. For example, we might like to maximize a profit given a certain initial investment, or we might wish to minimize the error produced by a computer simulation using a fixed number of CPUs. Thus, in such contexts, there are in general two components to such optimization problems:

- a set of constraints that must be satisfied
- an objective function to maximize or minimize subject to the given constraints.

In a business application, for instance, the constraints might include the amount of risk allowed in an investment portfolio, and in a scientific application, the constraints might be determined by the number of CPUs available to run a simulation. In either case, often the objective is to maximize profit or minimize cost.

An Example Optimization Problem

As a more detailed application, suppose that a web server company wants to buy new servers to replace outdated ones and has two options to choose from. There is a standard model which costs \$400, uses 300W of power, takes up two shelves of a server rack, and can handle 1000 hits/min. There is also a cutting-edge model, which costs \$1600, uses 500W of power, but takes up only one shelf, and can handle 2000 hits/min. With a budget of \$36,800, 44 shelves of server space and 12,200W of power, how many units of each model should the company purchase in order to maximize the number of hits it can serve every minute?

Let us introduce some variables, say x_1 and x_2, to represent the number of servers for each model. Then the number of hits per minute that can be serviced by x_1 standard servers and x_2 cutting-edge servers is

$$1000x_1 + 2000x_2.$$

Our goal is to maximize this quantity.

The number of servers the company should get is limited by three factors: the budget, which translates into

$$400x_1 + 1600x_2 \le 36800,$$

the number of shelves that can be taken up by these servers,

$$2x_1 + x_2 \le 44,$$

and the amount of power these servers can use collectively,

$$300x_1 + 500x_2 \le 12200.$$

Therefore, this optimization problem can be summarized as follows:

$$\begin{aligned}\text{maximize:}\quad & z = 1000x_1 + 2000x_2\\ \text{subject to:}\quad & 400x_1 + 1600x_2 \le 36800\\ & 2x_1 + x_2 \le 44\\ & 300x_1 + 500x_2 \le 12200\\ & x_1, x_2 \ge 0,\end{aligned}$$

where the inequalities in the last line express the implicit requirement that the number of each server has to be nonnegative. Such inequalities are necessary to prevent a nonsensical solution, and they are distinguished from the other constraints in that they determine the sign of the variables. Solving optimization problems that have the above general form is known as ***linear programming***, which is the topic we study in this chapter.

Linear programming encompasses a broad subclass of optimization problems, including the shortest-path problem and maximum flow, as well as fundamental applications from the realms of military, science, and business. Algorithms that solve linear programs (often abbreviated as LP) have been extensively studied in the last century. Our goal in this chapter is to understand one of these algorithms, which is still in use, called the simplex method.

Translating Problems into Linear Programs

In general, there are three steps for turning an optimization problem into a linear program, assuming such a formulation is possible:

1. Determining the variables of the problem.
2. Finding the quantity to optimize, and write it in terms of the variables.
3. Finding all the constraints on the variables and writing equations or inequalities to express these constraints.

In addition, in defining the constraints, we need to be sure to include any implicit constraints describing the range of values the variables can take and to make sure all the equations are ***linear***, as in the above example. In this chapter, we discuss how to create such formulations of optimization problems.

Once we have such a formulation, we then should solve the resulting linear program, and there are several software packages available for doing this that are based on various efficient algorithms. In this chapter, we focus on a classic algorithm for solving linear programs, which is known as the ***simplex method***. We also discuss an important topic known as ***duality***.

26.1 Formulating the Problem

Standard Form

Recall that a function, f, is a ***linear function*** in the variables, $x_1, x_2, \ldots, x_n$, if it has the following form:

$$f(x_1, x_2, \ldots, x_n) = a_1x_1 + a_2x_2 + \cdots + a_nx_n = \sum_{i=1}^{n} a_ix_i,$$

for some real numbers, $a_1, a_2, \ldots, a_n$, which are called ***coefficients*** or ***weights***.

A ***linear program*** in ***standard form*** is an optimization problem with the following form:

$$\begin{array}{ll} \text{maximize:} & z = \sum_{i \in V} c_ix_i \\ \text{subject to:} & \sum_{j \in V} a_{ij}x_j \leq b_i \text{ for } i \in C \\ & x_i \geq 0 \text{ for } i \in V \end{array}$$

where V indexes over the set of variables and C indexes over the set of constraints. The x_i's are variables, whereas all other symbols represent fixed real numbers. The function to maximize is called the ***objective function***, and the inequalities are called ***constraints***. In particular, the inequalities $x_i \geq 0$ are called ***nonnegativity constraints***. This program is linear because both the objective and the constraints are linear functions of the variables, where an inequality is linear if there is a linear function on one side and a constant on the other side of the inequality.

As an example, the earlier linear program is reproduced below. Notice that it fits the standard form. To make this fact more explicit, we rewrite it in gray using the notation from the definition:

$$\begin{array}{llr} \text{maximize:} & z = 1000x_1 + 2000x_2 & z = c_1x_1 + c_2x_2 \\ \text{subject to:} & 400x_1 + 1600x_2 \leq 36800 & a_{11}x_1 + a_{12}x_2 \leq b_1 \\ & 2x_1 + x_2 \leq 44 & a_{21}x_1 + a_{22}x_2 \leq b_2 \\ & 300x_1 + 500x_2 \leq 12200 & a_{31}x_1 + a_{32}x_2 \leq b_3 \\ & x_1, x_2 \geq 0 & x_1, x_2 \geq 0. \end{array}$$

There are two variables and three constraints, so $V = \{1, 2\}$ and $C = \{1, 2, 3\}$.

Linear programs also come in several variants of the standard form. For instance, we may want to minimize rather than maximize the objective function, the inequalities in the constraints may be expressed in terms of "greater than or equal to," or the constraints might be better expressed as equalities. Fortunately, these variants can easily be made to fit the standard form. The following table describes how this could be done.

The form …	can also be written as …
minimize $f(x_1,\ldots,x_n)$	maximize $-f(x_1,\ldots,x_n)$
$f(x_1,\ldots,x_n) \geq y$	$-f(x_1,\ldots,x_n) \leq -y$
$f(x_1,\ldots,x_n) = y$	$f(x_1,\ldots,x_n) \leq y$ $f(x_1,\ldots,x_n) \geq y$

For example, we can use the last two rules to rewrite $3x_1 - 2x_2 = 5$ into an equivalent form consisting of the two inequalities

$$\begin{aligned} 3x_1 - 2x_2 &\leq 5, \\ -3x_1 + 2x_2 &\leq -5. \end{aligned}$$

Matrix Notation

A linear function can be expressed as a dot product:

$$\sum_{i=1}^{n} a_i x_i = \vec{a} \cdot \vec{x}, \qquad \begin{aligned} \vec{a} &= (a_1,\ldots,a_n), \\ \vec{x} &= (x_1,\ldots,x_n). \end{aligned}$$

Notice that $\vec{a}$ is a vector of numbers while $\vec{x}$ is a vector of variables. Using dot products, we can express the standard form more compactly:

$$\begin{array}{ll} \text{maximize:} & \vec{c} \cdot \vec{x} \\ \text{subject to:} & \vec{a}_1 \cdot \vec{x} \leq b_1 \\ & \vec{a}_2 \cdot \vec{x} \leq b_2 \\ & \quad\vdots \\ & \vec{a}_m \cdot \vec{x} \leq b_m. \end{array}$$

In fact, we can express it even more compactly, by letting A be the matrix where the ith row is vector $\vec{a}_i$. In other words, A is the $m \times n$ matrix whose ijth entry is a_{ij}. Also let $\vec{b}$ be the vector with entries b_i. If we extend the meaning of the symbol $\leq$ to row-wise inequality, we can have a more succinct description of standard form as follows:

$$\begin{array}{ll} \text{maximize:} & \vec{c} \cdot \vec{x} \\ \text{subject to:} & A\vec{x} \leq \vec{b}. \end{array}$$

For example, the inequalities

$$\begin{aligned} 2x + z &\leq 5 \\ x - 4y - 3z &\leq 1 \end{aligned}$$

can be written as

$$\begin{aligned} (2,0,1) \cdot (x,y,z) &\leq 5 \\ (1,-4,-3) \cdot (x,y,z) &\leq 1 \end{aligned} \qquad \text{or} \qquad \begin{pmatrix} 2 & 0 & 1 \\ 1 & -4 & -3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leq \begin{pmatrix} 5 \\ 1 \end{pmatrix}.$$

The Geometry of Linear Programs

To understand a linear program, it helps to look at the problem from a geometric point of view. For simplicity, let us restrict ourselves to the two-dimensional case for the time being. In this case, each inequality constraint describes a half-plane, expressed as an infinite region to one side of a line. So a point that is inside all of these half-planes is a point that satisfies all the constraints. We call such a point a ***feasible solution***. Now intersections of half-planes have the shape of a convex polygon (Section 22.2). So the set of all feasible solutions is a convex polygon. We call this set the ***feasible region***. For example, Figure 26.1 shows the feasible region of the linear program from the web server example.

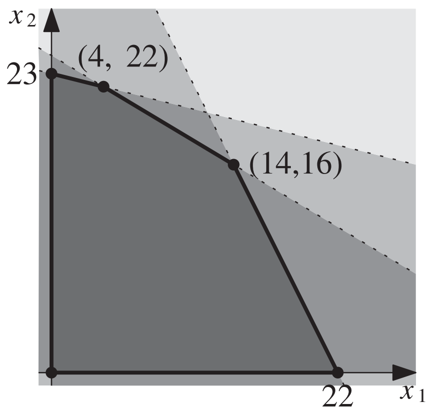


Figure 26.1: A feasible region is the intersection of half-spaces.

A region is ***convex*** if any two points in the region can be joined by a line segment entirely in the region (see Figure 26.2). Intuitively, if a two-dimensional shape is convex, then a person walking on its boundary, assuming it has one, would always be making left turns, or always right turns. (See, also, Section 22.2.)

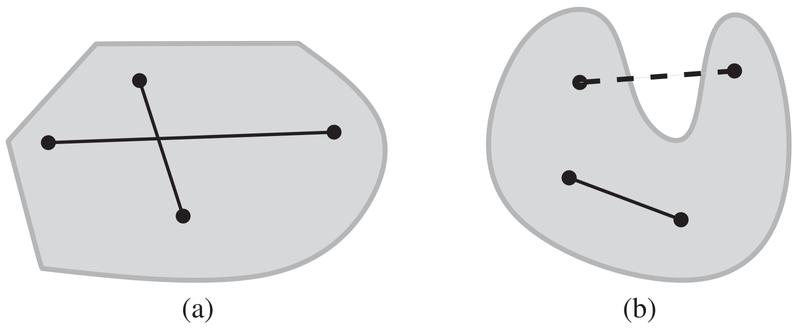


Figure 26.2: (a) In a convex set, any segment line joining two points inside the set is also inside the set. (b) In a non-convex set, there are segments joining two points in the set that are not entirely in the set.

In general, linear programs have a geometry in d dimensions where d is the number of variables. The geometric intuition is the same as in the plane but the shapes are more complex and the terminology is different. Inequality constraints in d dimensions are represented by half-spaces instead of half-planes, and their intersection forms a convex polytope instead of a convex polygon. For instance, a three-dimensional linear program could have a feasible region in the shape of a cube, pyramid, soccer ball, or any other three-dimensional convex shape with flat sides defining its boundary.

It is not enough to find just any feasible solution, of course. We are interested in one that optimizes the objective function. We refer to this feasible solution an ***optimal solution***. In the web server example, for instance, the set of points that produce a particular value, c, of the objective function is given by the equation

$$c = 1000x_1 + 2000x_2,$$

which is represented by a line. Such lines with varying values of c are all parallel, and we are interested in the one that maximizes c while still containing a feasible solution somewhere on the line. (See Figure 26.3.)

Referring to Figure 26.3, note that the slope of a line for the above objective function is always $-1/2$, regardless of the value of c. So we can imagine that, as c increases, the line sweeps the plane from the bottom left to the top right, staying parallel to itself. An optimal solution must be contained within the feasible region, which is represented by the gray region, and it must have the highest possible objective value. So the optimal solution is the point in the feasible region which is last hit by the sweeping line as it sweeps up and to the right. In this case, this point is the intersection of the lines representing the budget and power constraints,

$$400x_1 + 1600x_2 = 36800$$
$$300x_1 + 500x_2 = 12200.$$

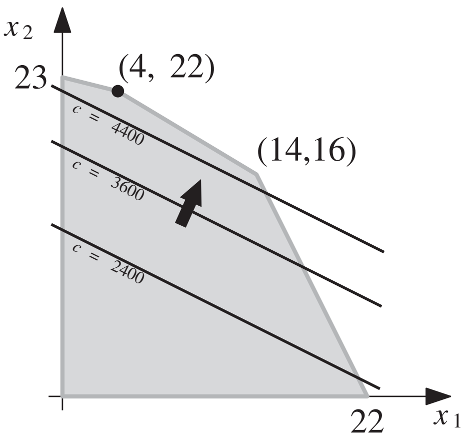


Figure 26.3: An example two-dimensional feasible region and objective function, $c = 1000x_1 + 2000x_2$. Values of the objective are represented by parallel lines.

By solving the above system of two equations in two variables, we see that the optimal solution occurs when $x_1 = 4$ and $x_2 = 22$. This example illustrates a principle that applies even to linear programs in higher dimensions—namely, that, because the feasible region is a convex polytope, an optimal solution, if one exists, always occurs on the boundary.

An optimal solution may not always be unique, however, such as when we modify the coefficients of the objective function in our example to

$$\text{maximize} \quad 1500x_1 + 2500x_2 .$$

(See Figure 26.4(a).) It is also possible that the solution does not exist at all. For example, this case occurs when the feasible region is unbounded and the objective function tends to $+\infty$ as we move along a ray contained in the feasible region. (See Figure 26.4(b).)

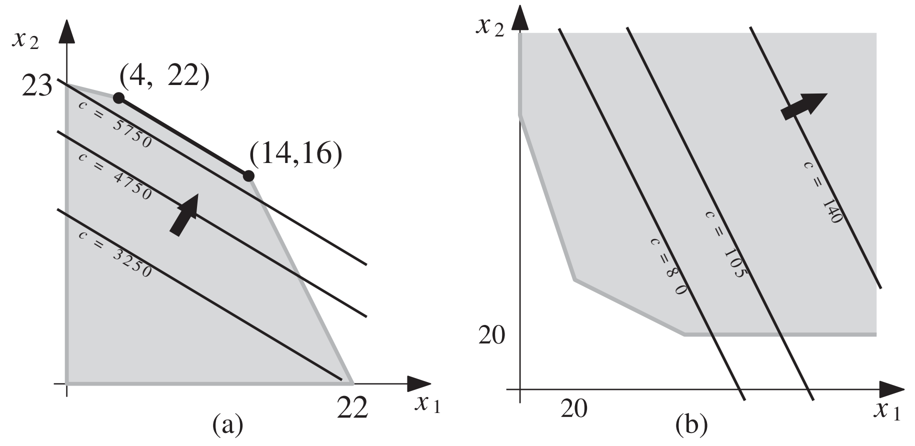


Figure 26.4: A linear program can have (a) many optimal solutions or (b) no optimal solution at all.

Finally, a linear program may not have an optimal solution simply because it has no feasible solution. This situation occurs when the constraints are so restrictive that no assignment to the variables can satisfy every constraint. Geometrically, this is the situation in which the intersection of all the half-spaces is empty.

Therefore, one algorithm to solve a linear program that has at least one optimal solution is to find all the vertices of the feasible region, and evaluate the objective function at these points. The optimal solution will be the point or points with highest objective value. This method is not particularly efficient, however, because there can be exponentially many vertices to evaluate. A better approach consists of starting at one vertex and, over several iterations, moving to a neighboring vertex with an increasingly better objective value. Thus, we can find a path on the boundary of the feasible region that starts at any vertex and ends at an optimal one. This alternative algorithm is called the simplex method, which is the algorithm we discuss next.

26.2 The Simplex Method

In this section, we describe the ***simplex method***, which is an algorithm for solving linear programs that follows a path through the vertices of the feasible region that increases the objective function in a greedy fashion. Although the worst-case runtime for this algorithm is exponential, in practice the algorithm usually finishes quickly.

26.2.1 Slack Form

To solve a linear program using the simplex method, we must first rewrite the problem in a format known as ***slack form***. To convert a linear program from standard form into slack form, we rewrite each of the inequality constraints as an equivalent equality constraint. This is done with the introduction of new variables, called ***slack variables***, which are nonnegative and measure the difference in the original inequality. For example, to rewrite the inequality $2x - 5y \leq 28$ in slack form, we could introduce a slack variable, s, with the constraints, $s = 28 - (2x - 5y)$ and $s \geq 0$. Intuitively, variable s measures the "slack" in the inequality, that is, the amount between the lesser and greater quantities in the inequality. We perform this step for each inequality in the standard form, introducing a slack variable for each such inequality.

Formally, we say that linear program is in ***slack form*** if we seek to maximize a linear objective, z, subject to constraints that are either equality constraints involving a slack variable or are nonnegativity constraints, as follows:

$$\begin{aligned}
&\text{maximize:} && z = c_* + \sum_{j \in F} c_j x_j \\
&\text{subject to:} && x_i = b_i - \sum_{j \in F} a_{ij} x_j, \quad \text{for } i \in B \\
& && x_i \geq 0 \text{ for } 1 \leq i \leq m + n.
\end{aligned}$$

The sets B and F partition the x_i variables into ***basic variables*** and ***free variables***, respectively. That is, each equality constraint has a basic (slack) variable on the left-hand side and only free variables on the right-hand side. Thus, free variables only appear on the left-hand side of nonnegativity constraints. Taken together, the a_{ij} coefficients form a $m \times n$ matrix, A, where $n = |F|$ is the number of variables in the standard form, and $m = |B|$ is the number of constraints in the standard form. Incidentally, the minus sign in the equality constraints is needed so that the matrix A is the same in the slack form and standard form.

Example 26.1: *Below we convert the linear program in standard form (left) into a slack form (right). In this particular slack form, the basic variables are slack variables, but this is not always the case.*

$$\begin{array}{rr@{\qquad}rl} \textit{maximize:} & z = x_1 + 2x_2 & \textit{maximize:} & z = x_1 + 2x_2 \\ \textit{subject to:} & -3x_1 + 2x_2 \le 3 & \textit{subject to:} & x_3 = 3 + 3x_1 - 2x_2 \\ & x_1 + x_2 \le 2 & & x_4 = 2 - x_1 - x_2 \\ & x_1 - x_2 \le 1 & & x_5 = 1 - x_1 + x_2 \\ & x_1, x_2 \ge 0 & & x_1, x_2, x_3, x_4, x_5 \ge 0 \end{array}$$

In this slack form, the free variables have indices $F = \{1, 2\}$ *and the basic variables have indices* $B = \{3, 4, 5\}$.

We are interested in the ***basic solution*** of the slack form, which means we set all the free variables to zero and let the equality constraints determine the values of the basic variables. In the above example, the basic solution is, $x_1 = x_2 = 0$, $x_3 = 3$, $x_4 = 2$, and $x_5 = 1$, or written in vector notation $\vec{x} = (0, 0, 3, 2, 1)$, and the objective function has value $z = c_* = 0$. In this case, the basic solution is a feasible solution, but this need not be the case in general.

The simplex method works by rewriting the slack form until a basic solution becomes an optimal solution. The operation we use to rewrite the slack form is called a ***pivot***, which takes a free variable and a basic variable and interchanges their roles by rewriting the equality constraints and objective function. Pivoting two variables produces an equivalent slack form, meaning that it has the same feasible region and that the objective function has the same values in the feasible region.

Example 26.2: *In this example, we perform a pivot where the free variable* x_1 *becomes basic and the basic variable* x_5 *becomes free. We perform this pivot by rewriting the equality constraint with* x_5 *on the left-hand side so that it has* x_1 *on the left-hand side. Then, we substitute this new equality constraint for* x_1 *in the old objective function and equality constraints to obtain a new objective function and new equality constraints involving only the free variables,* x_2 *and* x_5*, on the right-hand sides.*

$$\begin{array}{rl@{\qquad}rl} \textit{maximize:} & z = x_1 + 2x_2 & \textit{maximize:} & z = 1 + 3x_2 - x_5 \\ \textit{subject to:} & x_3 = 3 + 3x_1 - 2x_2 & \textit{subject to:} & x_3 = 6 + x_2 - x_5 \\ & x_4 = 2 - x_1 - x_2 & & x_4 = 2 - 2x_2 + x_5 \\ & x_5 = 1 - x_1 + x_2 & & x_1 = 1 + x_2 - x_5 \\ & x_1, x_2, x_3, x_4, x_5 \ge 0 & & x_1, x_2, x_3, x_4, x_5 \ge 0. \end{array}$$

The basic solution of the new slack form is $(1, 0, 6, 2, 0)$ *and the objective value is* 1. *So, as a result of the pivot, we have increased the objective value from* 0 *to* 1. *Geometrically, we have started from the vertex* $(0, 0)$*, where the inequalities* $x_1, x_2 \ge 0$ *are tight (that is, they are satisfied by equality with the right-hand side), and we have moved to the vertex* $(1, 0)$ *where the inequalities* $x_2, x_5 \ge 0$ *are tight.*

The simplex method describes the procedure for performing pivots that increase the objective value to the optimal objective value. Geometrically, pivots move from a vertex of the feasible region to a neighbor that increases the objective function the most. Thus, this approach is an application of the greedy method (Chapter 10).

Lemma 26.3: *The slack form of a LP is uniquely determined by the set of free variables.*

Proof: Suppose the two slack forms below are equivalent slack forms

$$\begin{aligned} \text{maximize:} \quad & z = c_* + \sum_{j \in F} c_j x_j \\ \text{subject to:} \quad & x_i = b_i - \sum_{j \in F} a_{ij} x_j \text{ for } i \in B \\ & x_i \geq 0 \text{ for } 1 \leq i \leq m+n \end{aligned}$$

$$\begin{aligned} \text{maximize:} \quad & z = c'_* + \sum_{j \in F} c'_j x_j \\ \text{subject to:} \quad & x_i = b'_i - \sum_{j \in F} a'_{ij} x_j \text{ for } i \in B \\ & x_i \geq 0 \text{ for } 1 \leq i \leq m+n \end{aligned}$$

We aim to show that they are in fact identical, i.e., $c_* = c'_*$, $c_j = c'_j$ for all $j \in F$, $b_i = b'_i$ for all $i \in B$, and $a_{ij} = a'_{ij}$ for all $i \in B$ and $j \in F$. Consider the equality

$$0 = \left(c_* + \sum_{j \in F} c_j x_j \right) - \left(c'_* + \sum_{j \in F} c'_j x_j \right) = (c_* - c'_*) + \sum_{j \in F} (c_j - c'_j) x_j.$$

This equation must hold for all feasible (nonnegative) values of the x_j. When $x_j = 0$ for all $j \in F$, we get $0 = c_* - c'_*$, so we must have that $c_* = c'_*$. Fix $r \in F$ then let $x_r = 1$ and $x_j = 0$ for $j \in F \setminus \{r\}$. This produces the equality $0 = c_r - c'_r$, which implies $c_r = c'_r$. Since r was arbitrary, we have that $c_j = c'_j$ for all $j \in F$.

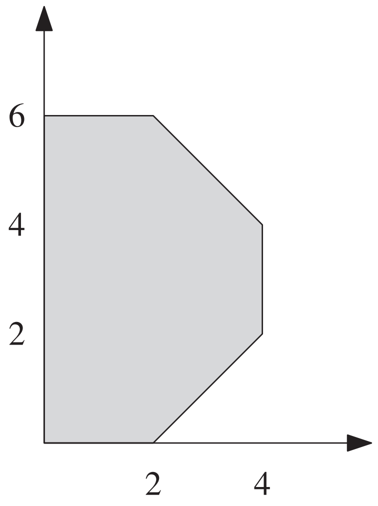
The remainder of the equalities are proved similarly, and left to Exercise C-26.14. ■

Earlier we said we would assume that the basic solution was feasible, but what do we do when this is not the case? If we know a vertex on the feasible region, then we can find which of n inequalities are tight there and pivot the associated variable so that they are free variables. Exercise C-26.5 asks you to show that the basic solution is feasible in this new slack form. This does not however help us to find a vertex on the feasible region, however, which will wait until we develop the details for the simplex method.

26.2.2 An Extended Example

In this section, we follow the approach of the simplex method to optimize the following linear program whose feasibly region is drawn to the right. The strategy will be to perform a sequence of pivot operations, which in the end will yield a slack form that is readily optimized.

$$\begin{aligned} \text{maximize:} \quad & z = 4x_1 + x_2 \\ \text{subject to:} \quad & x_2 \leq 6 \\ & x_1 + x_2 \leq 8 \\ & x_1 \leq 4 \\ & x_1 - x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$



First, we rewrite the linear program into the initial slack form, introducing the slack variables x_3, x_4, x_5 and x_6.

$$\begin{aligned} \text{maximize:} \quad & z = 4x_1 + x_2 \\ \text{subject to:} \quad & x_3 = 6 - x_2 \\ & x_4 = 8 - x_1 - x_2 \\ & x_5 = 4 - x_1 \\ & x_6 = 2 - x_1 + x_2 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{aligned}$$

Free Variables: $F = \{1, 2\}$
Basic Variables: $B = \{3, 4, 5, 6\}$
Basic Solution: $(0, 0, 6, 8, 4, 2)$
Objective Value: $c_* = 0$

Next, we look at the objective function and notice that increasing either x_1 or x_2 will increase the objective value. We choose to raise the value of x_1 as it has the largest coefficient. This greedy strategy does not necessarily improve runtime, by the way, but it is nevertheless often a useful choice in practice. While keeping x_2 fixed at zero, we increase x_1 as far as possible. The nonnegativity constraints of the basic variables impose the constraints $x_1 \leq \infty$ from x_3, $x_1 \leq 8$ from x_4, $x_1 \leq 4$ from x_5 and $x_1 \leq 2$ from x_6. The tightest of these constraints is $x_1 \leq 2$. So we pivot x_1 and x_6, setting $x_1 = 2$ and $x_6 = 0$ in the basic solution. This produces a new but equivalent slack form.

$$\begin{array}{ll} \text{maximize:} & z = 8 + 5x_2 - 4x_6 \\ \text{subject to:} & x_1 = 2 + x_2 - x_6 \\ & x_3 = 6 - x_2 \\ & x_4 = 6 - 2x_2 + x_6 \\ & x_5 = 2 - x_2 + x_6 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{array}$$

Free Variables: $F = \{2, 6\}$
Basic Variables: $B = \{1, 3, 4, 5\}$
Basic Solution: $(2, 0, 6, 6, 2, 0)$
Objective Value: $c_* = 8$

Since the coefficient on x_6 is negative, raising it from zero will only decrease the objective function. For this reason, we keep x_6 fixed at zero and raise x_2 until we hit the first constraint. The constraints on x_2 are: $x_2 \leq \infty$, $x_2 \leq 6$, $x_2 \leq 3$, and $x_2 \leq 2$. The tightest of these constraints is $x_2 \leq 2$ coming from the basic variable x_5. So we set $x_2 = 2$ and $x_5 = 0$ by pivoting x_2 and x_5. This yields a new slack form.

$$\begin{array}{ll} \text{maximize:} & z = 18 - 5x_5 + x_6 \\ \text{subject to:} & x_1 = 4 - x_5 \\ & x_2 = 2 - x_5 + x_6 \\ & x_3 = 4 + +x_5 - x_6 \\ & x_4 = 2 + 2x_5 - x_6 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{array}$$

Free Variables: $F = \{5, 6\}$
Basic Variables: $B = \{1, 2, 3, 4\}$
Basic Solution: $(4, 2, 4, 2, 0, 0)$
Objective Value: $c_* = 18$

In this slack form, the coefficient of x_5 is negative. So we keep x_5 fixed at zero and increase x_6. The tightest constraint is $x_6 \leq 2$ coming from the basic variable x_4. So we pivot x_4 and x_6, yielding the next slack form.

$$\begin{array}{ll} \text{maximize:} & z = 20 - 3x_5 - x_4 \\ \text{subject to:} & x_1 = 4 - x_5 \\ & x_2 = 4 - x_4 + x_5 \\ & x_3 = 2 + x_4 - x_5 \\ & x_6 = 2 + 2x_5 - x_4 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{array}$$

Free Variables: $F = \{4, 5\}$
Basic Variables: $B = \{1, 2, 3, 6\}$
Basic Solution: $(4, 4, 2, 0, 0, 2)$
Objective Value: $c_* = 20$

Now something interesting has happened! Both of the variables in the objective function have negative coefficients, so increasing either of them would decrease the objective function. So we conclude that we must be at the optimum and stop. From the basic solution we see that $x_1 = 4$, $x_2 = 4$ and $c_* = 20$, which is the solution to our original problem.

26.2.3 The Simplex Algorithm

We formalize the strategy used in the previous example into Algorithm 26.5, which is commonly referred to as the ***simplex method***. This version of the algorithm assumes it is given as input a slack form in which the basic solution is feasible. Earlier we showed how to produce such a slack form if we know a vertex of the feasible region, and in Exercise C-26.6 we explore how to find such a vertex.

```
SimplexMethod$(A, b, c, c_*, F, B)$ :
  while there exists $j \in F$ with $c_j > 0$ do
      $r \leftarrow \arg\max_{j \in F} c_j$
      for $i \in B$ do
          $k_i \leftarrow$ ( if $a_{ir} \neq 0$ then $-b_i/a_{ir}$ else $\infty$)
      $s \leftarrow \arg\min_{i \in B} k_i$
      if $k_s = \infty$ then
          return unbounded exception
      else
          Pivot $x_r$ and $x_s$.
  return $(A, b, c, c_*, F, B)$
```

Algorithm 26.5: The simplex method. We assume the input is given in slack form and that the basic solution is feasible.

Analysis of the Simplex Algorithm

To analyze the running time of the simplex method, we notice that each iteration of the loop considers a different slack form of the original problem. If a slack form appears more than once while running the simplex method, then the algorithm will cycle. Since there are only $\binom{n+m}{n}$ different slack forms, one for each choice of F, we know that the algorithm will either halt in $\binom{n+m}{n}$ steps or it will cycle. We can avoid cycling, however, by having an appropriate rule for choosing among optimal pivots that don't actually change the value of the objective function, which is a phenomenon known as ***degeneracy***. In practice, however, the simplex algorithm will often halt in polynomial time in any case.

Since the simplex algorithm only works for inputs whose basic solution is feasible, we need to take into account the amount of time required to put the problem in this form. In Exercise C-26.6, we explore how to find a vertex of the feasible region by optimizing a slack form where the basic solution is feasible, taking $\binom{n+m}{n}$ time. In addition, in Exercise C-26.5, we consider how to transform a slack form into a slack form in which the basic solution is feasible given a vertex on the feasible region in time $O(mn)$. Thus, the simplex method will optimize a linear program in at most $O(\binom{n+m}{n})$ time.

Example 26.4: *Consider the following linear program, written in standard form:*

$$\begin{aligned} &\textit{maximize:} && z = x_1 + 2x_2 \\ &\textit{subject to:} && -x_1 + x_2 \le 3 \\ &&& x_1 + 3x_2 \le 13 \\ &&& x_1 - x_2 \le 1 \end{aligned}$$

To solve this linear program using the simplex method, we first we rewrite the linear program in slack form, introducing the slack variables, x_3, x_4 and x_5.

$$\begin{aligned} &\textit{maximize:} && z = x_1 + 2x_2 \\ &\textit{subject to:} && x_3 = 3 + x_1 - x_2 \\ &&& x_4 = 13 - x_1 - 3x_2 \\ &&& x_5 = 1 - x_1 + x_2 \end{aligned}$$

We then choose to increase x_2, as it has the largest coefficient in the objective function. The most restrictive constraint is given by x_3. So we pivot x_2 and x_3, yielding the following new slack form with objective value, $c_* = 6$.

$$\begin{aligned} &\textit{maximize:} && z = 6 + 3x_1 - 2x_3 \\ &\textit{subject to:} && x_2 = 3 + x_1 - x_3 \\ &&& x_4 = 4 - 4x_1 + 3x_3 \\ &&& x_5 = 4 - x_3 \end{aligned}$$

Next, we increase x_1, as it has the largest coefficient in the objective function. The most restrictive constraint is x_4. So we pivot x_1 and x_4, which yields the following slack form with objective value, $c_* = 9$.

$$\begin{aligned} &\textit{maximize:} && z = 9 + 0.25x_3 - 0.25x_4 \\ &\textit{subject to:} && x_1 = 1 + 0.75x_3 - 0.25x_4 \\ &&& x_2 = 4 - 0.25x_3 - 0.25x_4 \\ &&& x_5 = 4 - x_3 \end{aligned}$$

This time, we increase x_3. Its most restrictive constraint comes from x_5. So we pivot x_3 with x_5. We get the new slack form below, with objective value, $c_* = 10$.

$$\begin{aligned} &\textit{maximize:} && z = 10 - 0.25x_4 - 0.25x_5 \\ &\textit{subject to:} && x_1 = 4 - 0.25x_4 - 0.75x_5 \\ &&& x_2 = 3 - 0.25x_4 + 0.25x_5 \\ &&& x_3 = 4 - x_5 \end{aligned}$$

Now that all the coefficients of the objective function are negative, we see that the optimal value for this linear program is 10, with $x_1 = 4$ and $x_2 = 3$.

26.3 Duality

To prove that Algorithm 26.5 does indeed provide the correct output, we must discuss a linear program related to the original problem called the ***dual***.

Suppose that our input LP has the standard form:

$$\begin{aligned}
&\text{maximize:} && z = \sum_{j \in V} c_j x_j \\
&\text{subject to:} && \sum_{j \in V} a_{ij} x_j \le b_i \text{ for } i \in C \\
& && x_j \ge 0 \text{ for } j \in V
\end{aligned}$$

It can be put into slack form as follows:

$$\begin{aligned}
&\text{maximize:} && z = \sum_{j \in F} c_j x_j \\
&\text{subject to:} && x_i = b_i - \sum_{j \in F} a_{ij} x_j \text{ for } i \in B \\
& && x_j \ge 0 \text{ for } j \in F \cup B
\end{aligned}$$

Therefore, $C = B$ and $V = F$. Note that this only holds because we did the obvious transformation between the two forms. There are many more slack forms equivalent to the original standard form, such as the slack forms produced at each iteration of Algorithm 26.5, whose indexing sets B and F do not directly correspond to C and V. Nevertheless, for this section, we assume that whenever we transform the standard form into slack form, we do the obvious transformation. So we can assume that $C = B$ and $V = F$.

Given this initial LP, the ***dual LP*** is a minimization problem which interchanges the roles of $\vec{b}$ and $\vec{c}$ and the roles of B and F. It also introduces new variables y_i:

$$\begin{aligned}
&\text{minimize:} && z = \sum_{i \in B} b_i y_i \\
&\text{subject to:} && \sum_{i \in B} a_{ij} y_i \ge c_j \text{ for } j \in F \\
& && y_i \ge 0 \text{ for } i \in B
\end{aligned}$$

When considering the original problem in relationship to its dual, we refer to the original as the ***primal*** LP. Note the symmetry between the primal and dual LPs written in matrix form:

$$\begin{array}{rll}
 & \text{maximize:} & z = \vec{c} \cdot \vec{x} \\
\text{primal:} & \text{subject to:} & A\vec{x} \le \vec{b} \\
 & & \vec{x} \ge \vec{0}
\end{array}
\qquad
\begin{array}{rll}
 & \text{minimize:} & z = \vec{b} \cdot \vec{y} \\
\text{dual:} & \text{subject to:} & A^t\vec{y} \ge \vec{c} \\
 & & \vec{y} \ge \vec{0}
\end{array}$$

When going from the standard to the slack form, we extend the set of original variables x_i (now called free) to a bigger set to include basic variables. Still, for this section, let us use $\vec{x}$ to refer to the original variables only, which are indexed by F.

Example 26.5: *Below is a primal LP written in standard form and its dual.*

$$\begin{array}{rr} \textit{maximize:} & z = x_1 + 2x_2 \\ \textit{subject to:} & -3x_1 + 2x_2 \le 3 \\ & x_1 + x_2 \le 2 \\ & x_1 - x_2 \le 1 \\ & x_1, x_2 \ge 0 \end{array} \qquad \begin{array}{rr} \textit{minimize:} & z = 3y_1 + 2y_2 + y_3 \\ \textit{subject to:} & -3y_1 + y_2 + y_3 \ge 1 \\ & 2y_1 + y_2 - y_3 \ge 2 \\ & y_1, y_2, y_3 \ge 0 \end{array}$$

Note that the dual is no longer in standard form, because of the inequality constraints. It can of course be easily converted (Exercise R-26.7).

As we shall see, the solutions to the primal and dual problems are closely related. Why is that the case? Let's go back to the web server example.

Example 26.6: *Recall the primal LP representing the web server problem.*

$$\begin{array}{lr} \textit{maximize:} & z = 1000x_1 + 2000x_2 \\ \textit{subject to:} & 400x_1 + 1600x_2 \le 36800 \\ & 2x_1 + x_2 \le 44 \\ & 300x_1 + 500x_2 \le 12200 \\ & x_1, x_2 \ge 0 \end{array}$$

The dual problem is:

$$\begin{array}{lr} \textit{minimize:} & z = 36800y_1 + 44y_2 + 12200y_3 \\ \textit{subject to:} & 400y_1 + 2y_2 + 300y_3 \ge 1000 \\ & 1600y_1 + y_2 + 500y_3 \ge 2000 \\ & y_1, y_2, y_3 \ge 0 \end{array}$$

There is a way to interpret this dual problem. Suppose a computer manufacturer claims that it can cater to the web server company's needs, and that, by doing so, it can offer a set of web servers that will outperform the two types of servers the web server company had planned on using, while staying within its resource limits.

To do so, the computer manufacturer must assess the number of hits/min each resource can potentially produce. To outperform the first type of server, it must build a web server that can handle at least 1000 hits/min if it costs \$400, occupies two shelves and uses 300W of power. This constraint corresponds to the first inequality of the dual LP. A similar explanation handles the case of the second type of servers.

The variables in this inequality represent the number of hits/min each unit of resource will contribute. For example, we can think of y_2 as the value of a single shelf measured in hits/mins, because a resource that contributes more hits/min is *likely to increase the price of the computer (this is obviously up for debate, but let us assume this is true to simplify things). Now the computer manufacturer is offering its services for profit. So it will try to minimize its own production costs given the amount of resources available. This production cost is the quantity described by the objective function of the dual LP.*

The minimum number of hits/min that the computer manufacturer can get away with offering is equal to the maximum number of hits/min its client can achieve by only using the two types of servers it had at its disposal.

Lemma 26.7: *Let $\vec{x}'$ be a feasible solution to the primal LP $(A, \vec{b}, \vec{c}, c_*, F, B)$ and let $\vec{y}'$ be a feasible solution to the dual LP. If $\vec{c}\cdot\vec{x}' = \vec{b}\cdot\vec{y}'$, then $\vec{x}'$ and $\vec{y}'$ are optimal for their respective problems.*

Proof: Any two solutions $\vec{x}$ and $\vec{y}$ that are feasible in the primal and dual problems respectively give us that

$$c_j \leq \sum_{i\in B} a_{ij}y_i \quad \text{and} \quad \sum_{j\in F} a_{ij}x_j \leq b_i.$$

Using these inequalities one at a time, we see that

$$\sum_{j\in F} c_jx_j \leq \sum_{j\in F}\sum_{i\in B} a_{ij}y_ix_j \leq \sum_{i\in B} b_iy_i.$$

So just by being feasible, we have $\vec{c}\cdot\vec{x} \leq \vec{b}\cdot\vec{y}$. In particular, $\vec{c}\cdot\vec{x} \leq \vec{b}\cdot\vec{y}'$, which means that the primal objective function cannot be larger than $\vec{b}\cdot\vec{y}'$ if we stay in the feasible region. Since this objective function achieves the value $\vec{b}\cdot\vec{y}'$ at $\vec{x}'$, this means $\vec{x}'$ is optimal for the primal problem. A similar argument shows that $\vec{y}'$ is optimal for the dual problem. ■

We now proceed to demonstrate the correctness of Algorithm 26.5.

Theorem 26.8: *Suppose that on input LP $(A, \vec{b}, \vec{c}, 0, F, B)$, Algorithm 26.5 returns the LP $(A', \vec{x}', \vec{c}', c_*, F', B')$, then $\vec{x}'$ is optimal for the input LP.*

Proof: We show that the vector

$$\vec{y} = (y_i, i \in B) \quad \text{defined by} \quad y_i = \begin{cases} -c_i' & i \in F' \cap B \\ 0 & i \in B' \cap B \end{cases}$$

is feasible for the dual LP and $\vec{c}\cdot\vec{x}' = \vec{b}\cdot\vec{y}$ and apply the previous lemma.

The input and output LPs are equivalent, so they have the same objective function:

$$\sum_{j\in F} c_jx_j = c_* + \sum_{i\in F'} c_i'x_i \tag{26.1}$$

$$= c_* + \sum_{i\in F'\cap F} c_i'x_i + \sum_{i\in F'\cap B} c_i'x_i. \tag{26.2}$$

The split sum comes from the fact that $F' \subseteq F \cup B$. Now the last sum can be rewritten as

$$\sum_{i \in F' \cap B} -y_i \left(b_i - \sum_{j \in F} a_{ij} x_j \right),$$

by using the definitions of y_i and rewriting the slack variables, x_i, such that $i \in F' \cap B \subseteq B$, in terms of the free variables, x_k, where $k \in F$. So we get

$$\sum_{j \in F} c_j x_j = \left(c_* - \sum_{i \in B} y_i b_i \right) + \sum_{j \in F} \left(c'_j + \sum_{i \in B} a_{ij} y_i \right) x_j,$$

where we have substituted the previous expression into the one before, rearranged the order of the terms, and summed over B and F instead of $F' \cap B$ and $F' \cap F$, because $c'_j = 0$ for all $j \notin F'$. Setting the coefficients on the left-hand side and right-hand side equal for each variable x_j as well as for the constant term (see Exercise C-26.15), we have that

$$c_j = c'_j + \sum_{i \in B} a_{ij} y_i, \quad j \in F \tag{26.3}$$

$$c_* = \sum_{i \in B} y_i b_i. \tag{26.4}$$

Note that because $c'_j \leq 0$ for any j (this is the terminating condition of Algorithm 26.5), Equation 26.3 becomes

$$c_j \leq \sum_{i \in B} a_{ij} y_i.$$

So $\vec{y}$ is feasible for the dual problem. Also note that if we plug $\vec{x}'$ into Equation 26.1,

$$\sum_{j \in F} c_j x'_j = c_* + \sum_{i \in F'} c'_i x'_i.$$

But since $x'_i = 0$ for $i \in F'$, we have that $\vec{c} \cdot \vec{x}' = c_*$. Combined with Equation 26.4, this shows that $\vec{c} \cdot \vec{x}' = \vec{b} \cdot \vec{y}$.

Thus, we have shown that (1) $\vec{y}$ is feasible and that (2) $\vec{c} \cdot \vec{x}' = \vec{b} \cdot \vec{y}$. These are the conditions needed for Lemma 26.7, which says that $\vec{x}'$ is optimal. ■

26.4 Applications of Linear Programming

Because linear programming has such a general formulation, it can be used to solve other algorithmic problems. In particular, if a problem can be expressed as a linear program, then we can design an algorithm to solve this problem by giving this linear program as the input to a linear program solver, such as the simplex method. In this section, we look at three familiar problems that can be reduced to linear programs.

Shortest Paths

In the shortest-path problem, for a specific source, s, and target, t, we are given a graph whose vertices represent locations, such that each edge, (u, v), has a weight, $d(u, v)$, that represents the distance between two locations, u and v. The goal is to find the shortest path between the source, s, and target, t.

We can formulate a linear program for this problem based on the setup and termination conditions for the Bellman-Ford algorithm (Section 14.3). We define for every vertex, v, a variable, d_v, which should represent the shortest distance between s and v. The initialization, $d_s = 0$, now becomes one of our constraints. To pinpoint the correct value for d_t, we use the termination condition of the Bellman-Ford algorithm—namely, that the triangle inequality holds for every edge. Recall that this is the condition that states that the shortest path from s to v should be no longer than the shortest path from s to any neighbor u of v followed by the path from u to v. We want d_t to be the largest value that satisfies these conditions, the same way the algorithm initializes $d_t = \infty$ and progressively decreases its value until all variables meet the termination condition. Thus, the corresponding linear program is the following:

$$\begin{array}{ll} \text{maximize:} & d_t \\ \text{subject to:} & d_s = 0 \\ & d_v \le d_u + d(u, v), \quad \text{for every edge, } (u, v) \end{array}$$

This is, of course, the variant of the shortest-paths problem that has a single source and single target. In the exercises, we explore modifications to this LP to accommodate generalizations of the shortest-path problem.

Network Flow

Recall from Chapter 16 that a flow network is a connected directed graph with a source and a sink, and in which each edge, e, has nonnegative weights called its capacity, $c(e)$.

A flow, f, is a new set of nonnegative edge weights that satisfies the following rules:

- ***The Capacity Rule:*** The flow through an edge must be less than the capacity of that edge
- ***The Conservation Rule:*** With the exception of the source and the sink, the flow into a vertex must equal the flow out of it.

The maximum flow problem is to find the maximum flow size, that is, the maximum amount of flow out of the source that satisfies the above two rules. This can be expressed with a linear program in which the variables are the edge weights, $f(e)$, the objective function is the flow size, and the constraints are given by the two rules making up the definition of a valid flow:

$$\begin{array}{lll}\text{maximize:} & \displaystyle\sum_{e\in E^+(s)} f(e) & \text{where } s \text{ is the source}\\ \text{subject to:} & 0 \le f(e) \le c(e) & \text{for all edges } e\\ & \displaystyle\sum_{e\in E^-(v)} f(e) = \sum_{e\in E^+(v)} f(e) & \text{for all vertices } v \text{ except for the source and the sink}\end{array}$$

Here, $E^+(v)$ and $E^-(v)$ represent the set of incoming and outgoing edges of a vertex v respectively. Note that if the capacities of a flow network are integers, the simplex algorithm will produce a solution with integer edge weights.

Maximum Matching

Suppose five children have gone to a dog shelter to find a new pet. As it happens, there are exactly five dogs up for adoption. Each child has a preference for certain breeds, whereas each dog has a preference for children of a certain temperament. Figure 26.6 shows a graph in which a child and a dog who like each other are connected by an edge.

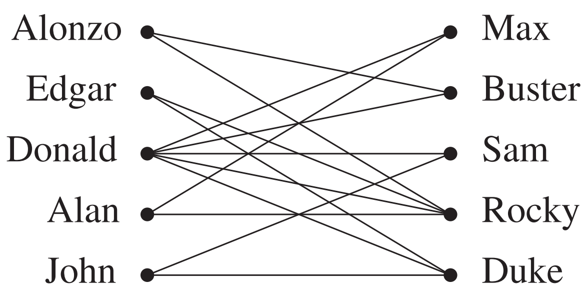


Figure 26.6: An edge connects a child and a dog who like each other.

Is there a pairing scheme in which all children and dogs will be happily matched? In other words, is there a maximum matching for the graph in Figure 26.6 equal to the number of children? In general, we can answer maximum matching questions by setting up a flow network, as done in Section 16.3, and then solving this problem using the LP given above. We create two additional vertices, a source s and a sink t. Then, we add a directed edge from the source to every child, and also an edge from every dog to the sink. Finally, we connect every child to a dog if they both like each other. We get the directed graph shown in Figure 26.7.

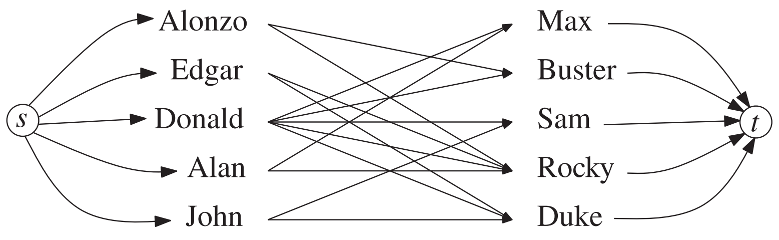


Figure 26.7: This flow network solves the bipartite matching problem.

Next, we assign a flow of 1 to each edge. Then there is a perfect matching in the original graph if and only if the maximum flow is equal to 5. Moreover, if we solve the flow network problem by converting it into an LP as we did earlier, all flows will be integers. This means that each edge can only have a flow of 0 or 1, so that we can interpret 0 as “do not pair them up” and 1 as “pair them up.” Thus, we can solving the maximum matching problem using a linear program solver.

26.5 Exercises

Reinforcement

R-26.1 Recall at the beginning of the chapter we gave a linear program to help a web server company decide what server models it should purchase. Suppose that the standard server model has been replaced by a new "green" server model, which only requires $200W$ of power, costs 600, takes up one shelf in the server rack, and can handle 800 hits/min. Give a new linear program that takes into account the availability of this new model.

R-26.2 Draw the two-dimensional feasible region of the LP from Exercise R-26.1.

R-26.3 Suppose that instead of maximizing hits per minute, constraints, a web server company wants to minimize cost while maintaining a rack of standard and cutting-edge servers that can handle at least 15,000 hits per minute. Also, for the sake of redundancy, the company wants to maintain at least 10 servers in their rack. Based on these constraints, give a linear program to find the optimal server configuration. Draw the feasible region, and solve the LP geometrically.

R-26.4 In the following linear program, the objective function has a parameter, α. What values of α result in a program with no unique solution?

$$\begin{array}{ll} \text{maximize:} & z = \alpha x_1 + x_2 \\ \text{subject to:} & 3x_1 + 5x_2 \leq 77 \\ & 7x_1 + 2x_2 \leq 56 \\ & x_1, x_2 \geq 0. \end{array}$$

R-26.5 Solve the linear program of Exercise R-26.4, for $\alpha = 1$, using the simplex method. Show the result of each pivot.

R-26.6 For each of the regions shown in Figure 26.8, give an LP for which that region is the feasible region, or explain why no such linear program exists.

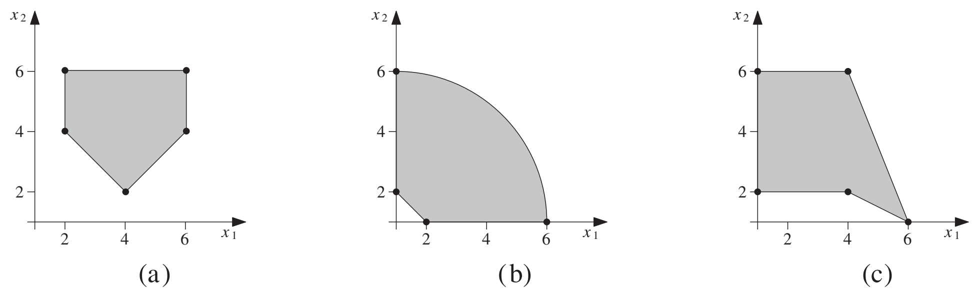


Figure 26.8: Different plausible feasible regions.

R-26.7 Convert the following linear program into standard form:

$$\begin{array}{ll} \text{minimize:} & z = 3y_1 + 2y_2 + y_3 \\ \text{subject to:} & -3y_1 + y_2 + y_3 \geq 1 \\ & 2y_1 + y_2 - y_3 \geq 2 \\ & y_1, y_2, y_3 \geq 0 \end{array}$$

R-26.8 Recall the LP for the dual of the web server problem from Example 26.6:

$$\begin{array}{ll} \text{minimize:} & z = 36800y_1 + 44y_2 + 12200y_3 \\ \text{subject to:} & 400y_1 + 2y_2 + 300y_3 \geq 1000 \\ & 1600y_1 + y_2 + 500y_3 \geq 2000 \\ & y_1, y_2, y_3 \geq 0 \end{array}$$

Convert this linear program first into standard, and then slack form.

R-26.9 Give a set of linear programming constraints that result in the feasible region shown in Figure 26.9.

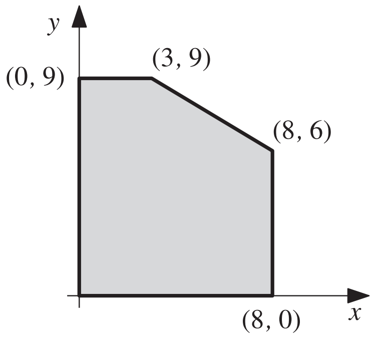


Figure 26.9: A feasible region.

R-26.10 For each vertex, $(3, 9)$ and $(8, 6)$, of the feasible region shown in Figure 26.9, give an objective function that has that vertex as the optimal solution.

R-26.11 Give an objective function for the feasible region shown in Figure 26.9, such that there are an infinite number of optimal solutions, none of which have $x = 0$ or $y = 0$. What is the value of the objective function for these solutions?

R-26.12 Formulate the dual of the linear program for the maximum flow problem.

R-26.13 What is the dual of the following linear program?

$$\begin{array}{ll} \text{maximize:} & z = x_1 + 2x_2 \\ \text{subject to:} & x_1 + x_2 \leq 5 \\ & 6x_1 - 3x_2 \leq 3 \\ & 5x_1 \leq 24 \\ & 6x_2 \leq 9 \\ & x_1, x_2 \geq 0 \end{array}$$

Creativity

C-26.1 Prove that there exists a linear program in two variables with exactly one feasible solution.

C-26.2 Prove that if there exists more than one optimal solution to a linear program, then there must be infinitely many optimal solutions.

C-26.3 Prove that the set of feasible solutions to a linear program with a nonempty feasible region is convex.

C-26.4 Give a linear program in three variables for which the feasible region is a tetrahedron.

C-26.5 When the simplex method was introduced, we assumed that the basic solution of the slack form was a feasible solution. Describe an algorithm that given an arbitrary slack form and a vertex on the feasible region, makes transformations to the slack form to guarantee that the basic solution is feasible.

C-26.6 Given a linear program in slack form such that the basic solution is feasible, give an algorithm to find a vertex of the feasible region by optimizing the slack form.

C-26.7 If P is a linear program, let P^* denote the dual of P, and let P^{k*} denote k application of the dual function. For example $P^{2*} = (P^*)^*$ is the dual of the dual of P. Show that for any positive integer n, $P^{2n*} = P$.

C-26.8 Show that if we allow linear programs to have strict inequalities, then there exists a linear program which is neither infeasible nor unbounded, but nevertheless does not have an optimal solution with finite objective value.

C-26.9 Prove that if there exists a point that is feasible in both a linear program and its dual, then that point is the optimal solution in both linear programs.

C-26.10 The maximum independent set (MIS) of a graph $G = (V, E)$ is the largest set of vertices $S \subseteq V$ such that for any two vertices $u, v \in S$, $(u, v) \notin E$; that is, no pair of vertices in S are neighbors. We want to create a linear program to solve the MIS problem. We create an indicator variable i_v for each vertex $v \in V$, and the sum of the i_v should denote the size of the MIS. Consider the following linear program formulation of the MIS problem:

$$\begin{array}{ll} \text{maximize:} & \sum i_v \\ \text{subject to:} & i_v \geq 0 \\ & i_v \leq 1 \\ & i_v + i_{v'} \leq 1 \text{ for } (v, v') \in E \end{array}$$

What's wrong with this formulation? Give a small example graph for which the linear program does not give a reasonable answer for the size of the maximum independent set. Why doesn't the program give a reasonable answer in this case?

C-26.11 In Section 26.4, we gave a linear program to solve the shortest-path problem. What can be said about this linear program if the input graph has a negative cycle?

C-26.12 Give a linear programming formulation for the all-pairs shortest-path problem.

C-26.13 Give a linear programming formulation to find the minimum spanning tree of a graph. Recall that a spanning tree T of a graph G is a connected acyclic subgraph of G that contains every vertex of G. The minimum spanning tree of a weighted graph G is a spanning tree T of G such that the sum of the edge weights in T is minimized.

C-26.14 Finish the proof of Lemma 26.3.

C-26.15 Finish the proof of Theorem 26.8 by showing that if

$$a_0 + a_1x_1 + a_2x_2 + \ldots a_nx_n = b_0 + b_1x_1 + b_2x_2 + \ldots b_nx_n,$$

where each x_i is a real variable, then $a_i = b_i$ for $i = 0, 1, \ldots n$.

Applications

A-26.1 In geometric data compression, we are given a collection of geometric objects, such as points in the plane, and we wish to minimize the number of bits required to represent the points. Therefore, suppose that rather than express the full (x, y) coordinates of such a set of points, we encode the y-coordinates of the points using the following rudimentary data compression scheme. First, we find a line $y = ax+b$ that approximately passes through the points. Then, instead of storing the full y-coordinate of each point, we store the function $f(x) = ax + b$ and we encode y-coordinate of each point (x_i, y_i) as $\epsilon_i = f(x_i) - y_i$. Suppose you are given the following set of points:

$$\{(0, 3), (-1.8, -6.1), (-4.2, -11), (1.5, 4.8), (6.3, 19), (2.6, 9.1)\}.$$

Give a linear program that, if solved, will find the function $f(x)$ minimizing the size of the encoded points using the above compression scheme.

A-26.2 Suppose that you are preparing for the upcoming Zombie Apocalypse. The Centers for Disease Control and Prevention recommend that any Zombie Apocalypse Survival Kit should contain at least the following supplies:

- Water (1 gallon per person per day)
- Food (stock up on nonperishable items that you eat regularly)
- Medications (this includes prescription and nonprescription meds)
- Tools and Supplies (utility knife, duct tape, battery-powered radio, etc.)
- Sanitation and Hygiene (household bleach, soap, towels, etc.)
- Clothing and Bedding (a change of clothes for each family member and blankets)
- Important documents (copies of your driver's license, passport, and birth certificate, to name a few)

- First Aid supplies (although you're a goner if a zombie bites you, you can use these supplies to treat basic cuts and lacerations)

Although not officially recommended by the CDC, your kit should also include at least one self-defense item to protect yourself, your family, and your supplies from zombies and looters. In case of evacuation, each person must be able to carry their survival kit with them. Therefore, your survival kit must not weigh more than 100 lbs. In addition, you must not spend more than $500 on your kit. Your goal is to assemble a kit that maximizes your life expectancy, subject to these weight and cost constraints. Choose reasonable values for the weight and cost of each category of item, and model how each category of item might affect your life expectancy. For example, each gallon of water might increase your life expectancy by 1 day. Then solve the corresponding linear program to determine which supplies you should include in your survival kit.

A-26.3 A political candidate has hired you to advise them on how to best spend their advertising budget. The candidate wants a combination of print, radio, and television ads that maximize total impact, subject to budgetary constraints, and available airtime and print space.

type	impact per ad	cost per ad	max ads per week
radio	a	10,000	25
print	b	70,000	7
tv	c	110,000	15

Design and solve a linear program to determine the best combination of ads for the campaign.

A-26.4 A perfect pizza maximizes how great it tastes and meets your recommended daily allowance (RDA) for the three macronutrients carbohydrates, fats, and protein. Suppose that your diet should consist of 25–35% fats, 45–65% carbohydrates, and 10–35% proteins. Suppose further that your total calorie intake should be is 1800–2500 calories. Given the following pizza ingredients, assign each pizza ingredient a "taste factor" per serving that represents your personal preferences. If your favorite ingredient is not included, look up its nutritional information and add it to the list of toppings. Design a linear program choose how much of each pizza topping you should include to make a pizza that meets the above constraints and maximizes how great it tastes. Explain how you chose your objective function and constraints.

Item		Calories	Fat	Carbs	Protein	Taste
Thin Crust	(per slice)	80	0.5g	15g	0g	*
Deep Dish		160	7.5	30g	0g	*
Pizza Sauce	(1 tbsp)	8	250mg	1g	250mg	*
Mozzarella	(10 g)	27	2g	298mg	3g	*
Pepperoni		46	4g	353mg	2g	*
Sausage		17	1g	707mg	1g	*
Green Pepper		2	17mg	464mg	86mg	*
Onion		4	10mg	934mg	110mg	*
Mushroom		2	34mg	328mg	309mg	*
Olive		11	1g	524mg	95mg	*
Pineapple		5	12mg	1g	54mg	*

A-26.5 A small retail chain has three warehouses and four retail stores. Each warehouse stores a certain amount of goods, and each retail store has a demand for a certain amount of goods. In addition, for each warehouse store pair, there is a set shipping cost per unit of goods. (See Figure 26.10.) Design and solve a linear program to determine the minimum-cost shipping schedule that meets the demands of each retail store.

warehouse supply	
W_1	100
W_2	75
W_3	120

retail demand	
A	40
B	30
C	50
D	45

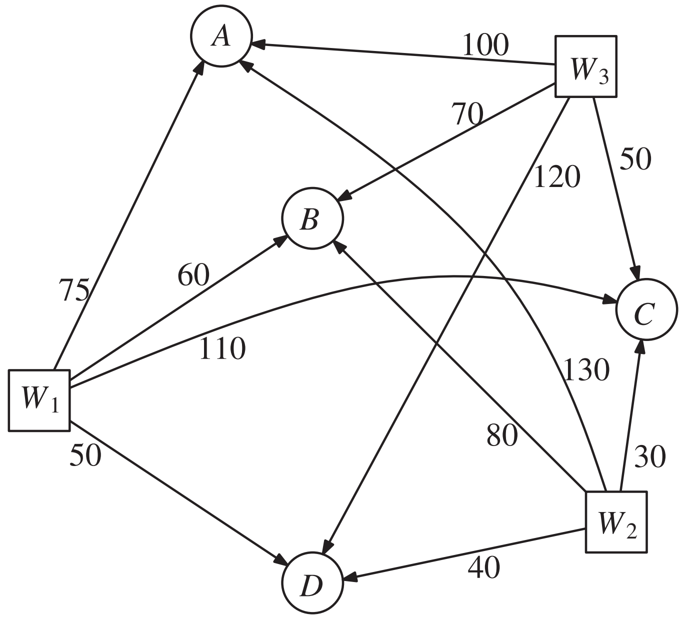


Figure 26.10: Shipping costs for warehouses and retail stores.

A-26.6 Suppose there are four power plants, which use coal, nuclear, wind, and oil, and four cities, Flat Top Mountain, Zephyrville, Cherenkov, and Iridescent Islands. The power plants and cities are connected by a directed power grid, and each link in the grid has a maximum capacity, as shown in Figure 26.11. Each city demands power with certain constraints. Flat Top Mountain demands 8 units of power, and insists that at least 5 units of this power be from the coal power plant. Zephyrville demands at least 5 units of power and insists that 4 units be generated from wind. Cherenkov demands 8 units of power and insists that 50% of its power be from nuclear power. Iridescent Islands demands at least 6 units of power but wants no more than 2 units of power from oil. Write a linear program to find the maximum flow of power that meets the demands of each city subject to the maximum capacities on the internal links in the power grid.

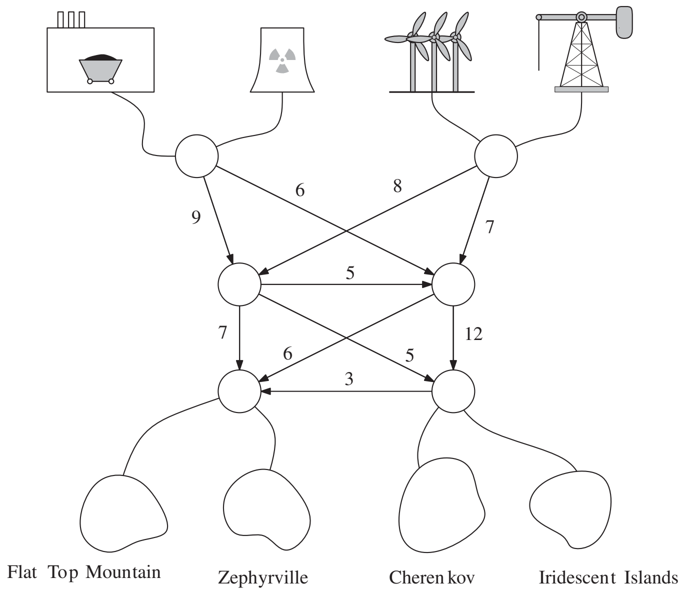


Figure 26.11: A coal, nuclear, and wind power plant power for four cities. Power must be routed through switches in the electrical grid.

A-26.7 Suppose you are part of a trade expedition and there are 15 people in your party (including yourself). Your final destination lies across the desert, and so you must hire out camels to carry all of your party's gear for that portion of your journey. The desert can be crossed in 10 days, but it could take up to 50% longer if something goes wrong or you encounter bad weather. You need to bring the following supplies necessary for survival in the harsh desert conditions:

- Some number of 18L jugs of water. You must bring at least 2L of water per person per day spent in the desert. Bring enough water to cross the desert even if something goes wrong.
- Enough tents such that at most two people will sleep in each tent. Each tent weighs 8kg each.
- A 1kg pack of first aid supplies per ten people-days spent in the desert (for example, 5 people for 4 days would require 2 first aid packs).
- Some cases of food rations. Each case weighs 10kg and contains 12 rations. Bring at least one ration per person per day spent in the desert.
- One pack per person of personal supplies (clothing, blankets, etc.) that weigh 20kg each.

In addition, you have the following trade goods: 5 chests of spices weighing 75kg each, 5 chests of tea weighing 50 kg each, and 5 chests of silk weighing 25kg each. Each camel can carry a 300kg load after accounting for the food, water, and gear required to support the camel over the long desert journey. Design a linear program to determine the minimum number of camels required to make the journey.

Chapter Notes

The word "programming" in "linear programming" does not have the same meaning as it has in computer science. It was first used in a mathematical sense by American mathematician, George B. Dantzig [54]. Linear programming was in use prior to this by the Soviet military, thanks to Leonid Kantorovich, who invented the concept in 1939. Today, linear programming is a standard tool of operations research, and it is used to model scheduling and assignment problems, as well as routing and planning problems.

The complexity of the simplex method puzzled mathematicians for many decades. While this algorithm seemed efficient in practice, carefully constructed problem instances can produce an exponential runtime. In 1980, Leonid Khachiyan published a new LP-solving algorithm [125] with a polynomial runtime that ran slow in practice. The ellipsoid algorithm, as it was called, was nevertheless revolutionary in concept. Unlike the simplex method, the ellipsoid algorithm walks through the interior of the feasible region. Narendra Karmarkar [121] eventually improved on this idea with what is now known as the interior point method, an algorithm that is fast in practice as well as having a polynomial runtime. The discrepancy between the theoretical and practical running times of the simplex method is explained by Spielman and Teng [199], who provide a new approach to analyze an algorithm's complexity in a way that is a cross between worst-case and average case analysis.

Appendix A Useful Mathematical Facts

In this appendix, we give several useful mathematical facts. We begin with some combinatorial definitions and facts.

Logarithms and Exponents

The logarithm function is defined as

$$\log_b a = c \qquad \text{if} \qquad a = b^c.$$

The following identities hold for logarithms and exponents, with $a, c > 0$:

1. $\log_b ac = \log_b a + \log_b c$
2. $\log_b a/c = \log_b a - \log_b c$
3. $\log_b a^c = c \log_b a$
4. $\log_b a = (\log_c a)/\log_c b$
5. $b^{\log_c a} = a^{\log_c b}$
6. $(b^a)^c = b^{ac}$
7. $b^a b^c = b^{a+c}$
8. $b^a/b^c = b^{a-c}$.

In addition, we have the following:

Theorem A.1: *If $a > 0$, $b > 0$, and $c > a + b$, then*

$$\log a + \log b \le 2 \log c - 2.$$

The ***natural logarithm*** function $\ln x = \log_e x$, where $e = 2.71828\ldots$, is the value of the following progression:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots .$$

In addition,

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

$$\ln(1 + x) = x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \cdots .$$

There are a number of useful inequalities relating to these functions (which derive from these definitions).

Theorem A.2: *If $x > -1$,*

$$\frac{x}{1+x} \le \ln(1+x) \le x.$$

Theorem A.3: *For $0 \le x < 1$,*

$$1 + x \le e^x \le \frac{1}{1-x}.$$

Theorem A.4: *For any two positive real numbers x and n,*

$$\left(1 + \frac{x}{n}\right)^n \le e^x \le \left(1 + \frac{x}{n}\right)^{n+x/2}.$$

Integer Functions and Relations

The "floor" and "ceiling" functions are defined respectively as follows:

1. $\lfloor x \rfloor =$ the largest integer less than or equal to x.
2. $\lceil x \rceil =$ the smallest integer greater than or equal to x.

The ***modulo*** operator is defined for integers $a \ge 0$ and $b > 0$ as

$$a \bmod b = a - \left\lfloor \frac{a}{b} \right\rfloor b.$$

The ***factorial*** function is defined as

$$n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (n-1)n.$$

The binomial coefficient is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!},$$

which is equal to the number of different ***combinations*** we can define by choosing k different items from a collection of n items (where the order does not matter). The name "binomial coefficient" derives from the ***binomial expansion***:

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}.$$

We also have the following relationships.

Theorem A.5: *If $0 \le k \le n$, then*

$$\left(\frac{n}{k}\right)^k \le \binom{n}{k} \le \frac{n^k}{k!}.$$

Theorem A.6 (Stirling's Approximation):

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \frac{1}{12n} + \epsilon(n)\right),$$

where $\epsilon(n)$ is $O(1/n^2)$.

The ***Fibonacci progression*** is a numeric progression such that $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$.

Theorem A.7: *If F_n is defined by the Fibonacci progression, then F_n is $\Theta(g^n)$, where $g = (1 + \sqrt{5})/2$ is the so-called* **golden ratio**.

Summations

There are a number of useful facts about summations.

Theorem A.8: *Factoring summations:*

$$\sum_{i=1}^{n} af(i) = a \sum_{i=1}^{n} f(i),$$

provided a does not depend upon i.

Theorem A.9: *Reversing the order:*

$$\sum_{i=1}^{n} \sum_{j=1}^{m} f(i,j) = \sum_{j=1}^{m} \sum_{i=1}^{n} f(i,j).$$

One special form of summation is a ***telescoping sum***:

$$\sum_{i=1}^{n} (f(i) - f(i-1)) = f(n) - f(0),$$

which often arises in the amortized analysis of a data structure or algorithm.

The following are some other facts about summations that often arise in the analysis of data structures and algorithms.

Theorem A.10:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

Theorem A.11:

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}.$$

Theorem A.12: *If $k \geq 1$ is an integer constant, then*

$$\sum_{i=1}^{n} i^k \text{ is } \Theta(n^{k+1}).$$

Another common summation is the ***geometric sum***

$$\sum_{i=0}^{n} a^i,$$

for any fixed real number $0 < a \neq 1$.

Theorem A.13:

$$\sum_{i=0}^{n} a^i = \frac{1-a^{n+1}}{1-a}$$

for any real number $0 < a \neq 1$.

Theorem A.14:

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1-a}$$

for any real number $0 < a < 1$.

There is also a combination of the two common forms, called the ***linear exponential*** summation, which has the following expansion:

Theorem A.15: *For* $0 < a \neq 1$, *and* $n \geq 2$,

$$\sum_{i=1}^{n} ia^i = \frac{a-(n+1)a^{(n+1)} + na^{(n+2)}}{(1-a)^2}.$$

The nth ***harmonic number*** H_n is defined as

$$H_n = \sum_{i=1}^{n} \frac{1}{i}.$$

Theorem A.16: *If* H_n *is the* n*th harmonic number, for* $n > 1$, *then* $\ln n < H_n < \ln n + 1$.

Proof: See Exercise C-3.11. ■

Useful Mathematical Techniques

To determine whether a function is little-oh or little-omega of another, it is sometimes helpful to apply the following rule.

Theorem A.17 (L'Hôpital's Rule): *If we have* $\lim_{n\to\infty} f(n) = +\infty$ *and we have* $\lim_{n\to\infty} g(n) = +\infty$, *then* $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} f'(n)/g'(n)$, *where* $f'(n)$ *and* $g'(n)$ *denote the derivatives of* $f(n)$ *and* $g(n)$, *respectively.*

In deriving an upper or lower bound for a summation, it is often useful to ***split a summation*** as follows:

$$\sum_{i=1}^{n} f(i) = \sum_{i=1}^{j} f(i) + \sum_{i=j+1}^{n} f(i).$$

Another useful technique is to ***bound a sum by an integral***. If f is a nondecreasing function, then, assuming the following terms are defined,

$$\int_{a-1}^{b} f(x)\,dx \leq \sum_{i=a}^{b} f(i) \leq \int_{a}^{b+1} f(x)\,dx.$$