29 Linear Programming

Many problems take the form of maximizing or minimizing an objective, given limited resources and competing constraints. If you can specify the objective as a linear function of certain variables, and if you can specify the constraints on resources as equalities or inequalities on those variables, then you have a ***linear-programming problem***. Linear programs arise in a variety of practical applications. We begin by studying an application in electoral politics.

A political problem

Suppose that you are a politician trying to win an election. Your district has three different types of areas—urban, suburban, and rural. These areas have, respectively, 100,000, 200,000, and 50,000 registered voters. Although not all the registered voters actually go to the polls, you decide that to govern effectively, you would like at least half the registered voters in each of the three regions to vote for you. You are honorable and would never consider supporting policies you don't believe in. You realize, however, that certain issues may be more effective in winning votes in certain places. Your primary issues are preparing for a zombie apocalypse, equipping sharks with lasers, building highways for flying cars, and allowing dolphins to vote.

According to your campaign staff's research, you can estimate how many votes you win or lose from each population segment by spending $1,000 on advertising on each issue. This information appears in the table of Figure 29.1. In this table, each entry indicates the number of thousands of either urban, suburban, or rural voters who would be won over by spending $1,000 on advertising in support of a particular issue. Negative entries denote votes that would be lost. Your task is to figure out the minimum amount of money that you need to spend in order to win 50,000 urban votes, 100,000 suburban votes, and 25,000 rural votes.

You could, by trial and error, devise a strategy that wins the required number of votes, but the strategy you come up with might not be the least expensive one. For example, you could devote $20,000 of advertising to preparing for a zombie

policy	urban	suburban	rural
zombie apocalypse	−2	5	3
sharks with lasers	8	2	−5
highways for flying cars	0	0	10
dolphins voting	10	0	−2

Figure 29.1 The effects of policies on voters. Each entry describes the number of thousands of urban, suburban, or rural voters who could be won over by spending $1,000 on advertising support of a policy on a particular issue. Negative entries denote votes that would be lost.

apocalypse, $0 to equipping sharks with lasers, $4,000 to building highways for flying cars, and $9,000 to allowing dolphins to vote. In this case, you would win $(20 \cdot -2) + (0 \cdot 8) + (4 \cdot 0) + (9 \cdot 10) = 50$ thousand urban votes, $(20 \cdot 5) + (0 \cdot 2) + (4 \cdot 0) + (9 \cdot 0) = 100$ thousand suburban votes, and $(20 \cdot 3) + (0 \cdot -5) + (4 \cdot 10) + (9 \cdot -2) = 82$ thousand rural votes. You would win the exact number of votes desired in the urban and suburban areas and more than enough votes in the rural area. (In fact, according to your model, in the rural area you would receive more votes than there are voters.) In order to garner these votes, you would have paid for $20 + 0 + 4 + 9 = 33$ thousand dollars of advertising.

It's natural to wonder whether this strategy is the best possible. That is, can you achieve your goals while spending less on advertising? Additional trial and error might help you to answer this question, but a better approach is to formulate (or ***model***) this question mathematically.

The first step is to decide what decisions you have to make and to introduce variables that capture these decisions. Since you have four decisions, you introduce four ***decision variables***:

- x_1 is the number of thousands of dollars spent on advertising on preparing for a zombie apocalypse,
- x_2 is the number of thousands of dollars spent on advertising on equipping sharks with lasers,
- x_3 is the number of thousands of dollars spent on advertising on building highways for flying cars, and
- x_4 is the number of thousands of dollars spent on advertising on allowing dolphins to vote.

You then think about ***constraints***, which are limits, or restrictions, on the values that the decision variables can take. You can write the requirement that you win at least 50,000 urban votes as

$$-2x_1 + 8x_2 + 0x_3 + 10x_4 \geq 50\,. \tag{29.1}$$

Similarly, you can write the requirements that you win at least 100,000 suburban votes and 25,000 rural votes as

$$5x_1 + 2x_2 + 0x_3 + 0x_4 \geq 100 \tag{29.2}$$

and

$$3x_1 - 5x_2 + 10x_3 - 2x_4 \geq 25 \; . \tag{29.3}$$

Any setting of the variables x_1, x_2, x_3, x_4 that satisfies inequalities (29.1)–(29.3) yields a strategy that wins a sufficient number of each type of vote.

Finally, you think about your ***objective***, which is the quantity that you wish to either minimize or maximize. In order to keep costs as small as possible, you would like to minimize the amount spent on advertising. That is, you want to minimize the expression

$$x_1 + x_2 + x_3 + x_4 \; . \tag{29.4}$$

Although negative advertising often occurs in political campaigns, there is no such thing as negative-cost advertising. Consequently, you require that

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, \text{ and } x_4 \geq 0 \; . \tag{29.5}$$

Combining inequalities (29.1)–(29.3) and (29.5) with the objective of minimizing (29.4) produces what is known as a "linear program." We can format this problem tabularly as

$$\begin{aligned}
&\text{minimize} && x_1 + x_2 + x_3 + x_4 && && && && && && (29.6)\\
&\text{subject to} \\
& && -2x_1 + 8x_2 + 0x_3 + 10x_4 && \geq && 50 && && && (29.7)\\
& && 5x_1 + 2x_2 + 0x_3 + 0x_4 && \geq && 100 && && && (29.8)\\
& && 3x_1 - 5x_2 + 10x_3 - 2x_4 && \geq && 25 && && && (29.9)\\
& && x_1, x_2, x_3, x_4 && \geq && 0 && . && && (29.10)
\end{aligned}$$

The solution to this linear program yields your optimal strategy.

The remainder of this chapter covers how to formulate linear programs and is an introduction to modeling in general. Modeling refers to the general process of converting a problem into a mathematical form amenable to solution by an algorithm. Section 29.1 discusses briefly the algorithmic aspects of linear programming, although it does not include the details of a linear-programming algorithm. Throughout this book, we have seen ways to model problems, such as by shortest paths and connectivity in a graph. When modeling a problem as a linear program, you go through the steps used in this political example—identifying the decision variables, specifying the constraints, and formulating the objective function. In order to model a problem as a linear program, the constraints and objectives must be

linear. In Section 29.2, we will see several other examples of modeling via linear programs. Section 29.3 discusses duality, an important concept in linear programming and other optimization algorithms.

29.1 Linear programming formulations and algorithms

Linear programs take a particular form, which we will examine in this section. Multiple algorithms have been developed to solve linear programs. Some run in polynomial time, some do not, but they are all too complicated to show here. Instead, we will give an example that demonstrates some ideas behind the simplex algorithm, which is currently the most commonly deployed solution method.

General linear programs

In the general linear-programming problem, we wish to optimize a linear function subject to a set of linear inequalities. Given a set of real numbers $a_1, a_2, \ldots, a_n$ and a set of variables $x_1, x_2, \ldots, x_n$, we define a ***linear function*** f on those variables by

$$f(x_1, x_2, \ldots, x_n) = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = \sum_{j=1}^{n} a_j x_j \ .$$

If b is a real number and f is a linear function, then the equation

$$f(x_1, x_2, \ldots, x_n) = b$$

is a ***linear equality*** and the inequalities

$$f(x_1, x_2, \ldots, x_n) \le b \quad \text{and} \quad f(x_1, x_2, \ldots, x_n) \ge b$$

are ***linear inequalities***. We use the general term ***linear constraints*** to denote either linear equalities or linear inequalities. Linear programming does not allow strict inequalities. Formally, a ***linear-programming problem*** is the problem of either minimizing or maximizing a linear function subject to a finite set of linear constraints. If minimizing, we call the linear program a ***minimization linear program***, and if maximizing, we call the linear program a ***maximization linear program***.

In order to discuss linear-programming algorithms and properties, it will be helpful to use a standard notation for the input. By convention, a maximization linear program takes as input n real numbers $c_1, c_2, \ldots, c_n$; m real numbers $b_1, b_2, \ldots, b_m$; and mn real numbers a_{ij} for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

The goal is to find n real numbers $x_1, x_2, \ldots, x_n$ that

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j \tag{29.11}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i \quad \text{for } i = 1, 2, \ldots, m \tag{29.12}$$

$$x_j \ge 0 \quad \text{for } j = 1, 2, \ldots, n \ . \tag{29.13}$$

We call expression (29.11) the ***objective function*** and the $n + m$ inequalities in lines (29.12) and (29.13) the ***constraints***. The n constraints in line (29.13) are the ***nonnegativity constraints***. It can sometimes be more convenient to express a linear program in a more compact form. If we create an $m \times n$ matrix $A = (a_{ij})$, an m-vector $b = (b_i)$, an n-vector $c = (c_j)$, and an n-vector $x = (x_j)$, then we can rewrite the linear program defined in (29.11)–(29.13) as

$$\text{maximize} \quad c^{\mathrm{T}} x \tag{29.14}$$

subject to

$$Ax \le b \tag{29.15}$$

$$x \ge 0 \ . \tag{29.16}$$

In line (29.14), $c^{\mathrm{T}} x$ is the inner product of two n-vectors. In inequality (29.15), Ax is the m-vector that is the product of an $m \times n$ matrix and an n-vector, and in inequality (29.16), $x \ge 0$ means that each entry of the vector x must be nonnegative. We call this representation the ***standard form*** for a linear program, and we adopt the convention that A, b, and c always have the dimensions given above.

The standard form above may not naturally correspond to real-life situations you are trying to model. For example, you might have equality constraints or variables that can take on negative values. Exercises 29.1-6 and 29.1-7 ask you to show how to convert any linear program into this standard form.

We now introduce terminology to describe solutions to linear programs. We denote a particular setting of the values in a variable, say x, by putting a bar over the variable name: $\bar{x}$. If $\bar{x}$ satisfies all the constraints, then it is a ***feasible solution***, but if it fails to satisfy at least one constraint, then it is an ***infeasible solution***. We say that a solution $\bar{x}$ has ***objective value*** $c^{\mathrm{T}} \bar{x}$. A feasible solution $\bar{x}$ whose objective value is maximum over all feasible solutions is an ***optimal solution***, and we call its objective value $c^{\mathrm{T}} \bar{x}$ the ***optimal objective value***. If a linear program has no feasible solutions, we say that the linear program is ***infeasible***, and otherwise, it is ***feasible***. The set of points that satisfy all the constraints is the ***feasible region***. If a linear program has some feasible solutions but does not have a finite optimal objective value, then the feasible region is ***unbounded*** and so is the linear program. Exercise 29.1-5 asks you to show that a linear program can have a finite optimal objective value even if the feasible region is unbounded.

One of the reasons for the power and popularity of linear programming is that linear programs can, in general, be solved efficiently. There are two classes of algorithms, known as ellipsoid algorithms and interior-point algorithms, that solve linear programs in polynomial time. In addition, the simplex algorithm is widely used. Although it does not run in polynomial time in the worst case, it tends to perform well in practice.

We will not give a detailed algorithm for linear programming, but will discuss a few important ideas. First, we will give an example of using a geometric procedure to solve a two-variable linear program. Although this example does not immediately generalize to an efficient algorithm for larger problems, it introduces some important concepts for linear programming and for optimization in general.

A two-variable linear program

Let us first consider the following linear program with two variables:

$$\begin{aligned}
&\text{maximize} && x_1 + x_2 && && && && (29.17)\\
&\text{subject to} && && && && && \\
& && 4x_1 - x_2 && \le 8 && && && (29.18)\\
& && 2x_1 + x_2 && \le 10 && && && (29.19)\\
& && 5x_1 - 2x_2 && \ge -2 && && && (29.20)\\
& && x_1, x_2 && \ge 0 \ . && && && (29.21)
\end{aligned}$$

Figure 29.2(a) graphs the constraints in the (x_1, x_2)-Cartesian coordinate system. The feasible region in the two-dimensional space (highlighted in blue in the figure) is convex.[1] Conceptually, you could evaluate the objective function $x_1 + x_2$ at each point in the feasible region, and then identify a point that has the maximum objective value as an optimal solution. For this example (and for most linear programs), however, the feasible region contains an infinite number of points, and so to solve this linear program, you need an efficient way to find a point that achieves the maximum objective value without explicitly evaluating the objective function at every point in the feasible region.

In two dimensions, you can optimize via a graphical procedure. The set of points for which $x_1 + x_2 = z$, for any z, is a line with a slope of -1. Plotting $x_1 + x_2 = 0$ produces the line with slope -1 through the origin, as in Figure 29.2(b). The intersection of this line and the feasible region is the set of feasible solutions that have an objective value of 0. In this case, that intersection of the line with the feasible region is the single point $(0, 0)$. More generally, for any value z, the

[1] An intuitive definition of a convex region is that it fulfills the requirement that for any two points in the region, all points on a line segment between them are also in the region.

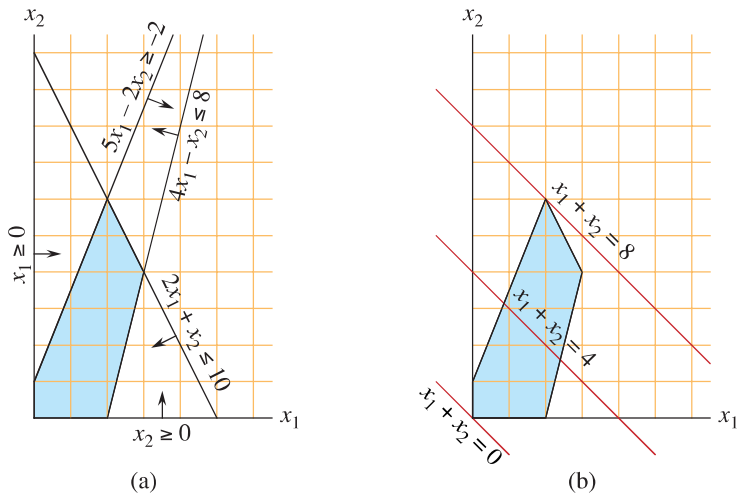

Figure 29.2 **(a)** The linear program given in (29.18)–(29.21). Each constraint is represented by a line and a direction. The intersection of the constraints, which is the feasible region, is highlighted in blue. **(b)** The red lines show, respectively, the points for which the objective value is 0, 4, and 8. The optimal solution to the linear program is $x_1 = 2$ and $x_2 = 6$ with objective value 8.

intersection of the line $x_1 + x_2 = z$ and the feasible region is the set of feasible solutions that have objective value z. Figure 29.2(b) shows the lines $x_1 + x_2 = 0$, $x_1 + x_2 = 4$, and $x_1 + x_2 = 8$. Because the feasible region in Figure 29.2 is bounded, there must be some maximum value z for which the intersection of the line $x_1 + x_2 = z$ and the feasible region is nonempty. Any point in the feasible region that maximizes $x_1 + x_2$ is an optimal solution to the linear program, which in this case is the vertex of the feasible region at $x_1 = 2$ and $x_2 = 6$, with objective value 8.

It is no accident that an optimal solution to the linear program occurs at a vertex of the feasible region. The maximum value of z for which the line $x_1 + x_2 = z$ intersects the feasible region must be on the boundary of the feasible region, and thus the intersection of this line with the boundary of the feasible region is either a single vertex or a line segment. If the intersection is a single vertex, then there is just one optimal solution, and it is that vertex. If the intersection is a line segment, every point on that line segment must have the same objective value. In particular, both endpoints of the line segment are optimal solutions. Since each endpoint of a line segment is a vertex, there is an optimal solution at a vertex in this case as well.

Although you cannot easily graph linear programs with more than two variables, the same intuition holds. If you have three variables, then each constraint corresponds to a half-space in three-dimensional space. The intersection of these half-

spaces forms the feasible region. The set of points for which the objective function obtains a given value z is now a plane (assuming no degenerate conditions). If all coefficients of the objective function are nonnegative, and if the origin is a feasible solution to the linear program, then as you move this plane away from the origin, in a direction normal to the objective function, you find points of increasing objective value. (If the origin is not feasible or if some coefficients in the objective function are negative, the intuitive picture becomes slightly more complicated.) As in two dimensions, because the feasible region is convex, the set of points that achieve the optimal objective value must include a vertex of the feasible region. Similarly, if you have n variables, each constraint defines a half-space in n-dimensional space. We call the feasible region formed by the intersection of these half-spaces a ***simplex***. The objective function is now a hyperplane and, because of convexity, an optimal solution still occurs at a vertex of the simplex. Any algorithm for linear programming must also identify linear programs that have no solutions, as well as linear programs that have no finite optimal solution.

The ***simplex algorithm*** takes as input a linear program and returns an optimal solution. It starts at some vertex of the simplex and performs a sequence of iterations. In each iteration, it moves along an edge of the simplex from a current vertex to a neighboring vertex whose objective value is no smaller than that of the current vertex (and usually is larger.) The simplex algorithm terminates when it reaches a local maximum, which is a vertex from which all neighboring vertices have a smaller objective value. Because the feasible region is convex and the objective function is linear, this local optimum is actually a global optimum. In Section 29.3, we'll see an important concept called "duality," which we'll use to prove that the solution returned by the simplex algorithm is indeed optimal.

The simplex algorithm, when implemented carefully, often solves general linear programs quickly in practice. With some carefully contrived inputs, however, the simplex algorithm can require exponential time. The first polynomial-time algorithm for linear programming was the ***ellipsoid algorithm***, which runs slowly in practice. A second class of polynomial-time algorithms are known as ***interior-point methods***. In contrast to the simplex algorithm, which moves along the exterior of the feasible region and maintains a feasible solution that is a vertex of the simplex at each iteration, these algorithms move through the interior of the feasible region. The intermediate solutions, while feasible, are not necessarily vertices of the simplex, but the final solution is a vertex. For large inputs, interior-point algorithms can run as fast as, and sometimes faster than, the simplex algorithm. The chapter notes point you to more information about these algorithms.

If you add to a linear program the additional requirement that all variables take on integer values, you have an ***integer linear program***. Exercise 34.5-3 on page 1098 asks you to show that just finding a feasible solution to this problem is NP-hard. Since no polynomial-time algorithms are known for any NP-hard prob-

lems, there is no known polynomial-time algorithm for integer linear programming. In contrast, a general linear-programming problem can be solved in polynomial time.

Exercises

29.1-1

Consider the linear program

$$\begin{aligned}
&\text{minimize} && -2x_1 + 3x_2 \\
&\text{subject to} \\
&&& x_1 + x_2 = 7 \\
&&& x_1 - 2x_2 \le 4 \\
&&& x_1 \ge 0 \ .
\end{aligned}$$

Give three feasible solutions to this linear program. What is the objective value of each one?

29.1-2

Consider the following linear program, which has a nonpositivity constraint:

$$\begin{aligned}
&\text{minimize} && 2x_1 + 7x_2 + x_3 \\
&\text{subject to} \\
&&& x_1 - x_3 = 7 \\
&&& 3x_1 + x_2 \ge 24 \\
&&& x_2 \ge 0 \\
&&& x_3 \le 0 \ .
\end{aligned}$$

Give three feasible solutions to this linear program. What is the objective value of each one?

29.1-3

Show that the following linear program is infeasible:

$$\begin{aligned}
&\text{maximize} && 3x_1 - 2x_2 \\
&\text{subject to} \\
&&& x_1 + x_2 \le 2 \\
&&& -2x_1 - 2x_2 \le -10 \\
&&& x_1, x_2 \ge 0 \ .
\end{aligned}$$

29.1-4

Show that the following linear program is unbounded:

$$\begin{aligned}
\text{maximize} \quad & x_1 - x_2 \\
\text{subject to} \quad & \\
& -2x_1 + x_2 \leq -1 \\
& -x_1 - 2x_2 \leq -2 \\
& x_1, x_2 \geq 0 \ .
\end{aligned}$$

29.1-5

Give an example of a linear program for which the feasible region is not bounded, but the optimal objective value is finite.

29.1-6

Sometimes, in a linear program, you need to convert constraints from one form to another.

a. Show how to convert an equality constraint into an equivalent set of inequalities. That is, given a constraint $\sum_{j=1}^{n} a_{ij} x_j = b_i$, give a set of inequalities that will be satisfied if and only if $\sum_{j=1}^{n} a_{ij} x_j = b_i$,

b. Show how to convert an inequality constraint $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ into an equality constraint and a nonnegativity constraint. You will need to introduce an additional variable s, and use the constraint that $s \geq 0$.

29.1-7

Explain how to convert a minimization linear program to an equivalent maximization linear program, and argue that your new linear program is equivalent to the original one.

29.1-8

In the political problem at the beginning of this chapter, there are feasible solutions that correspond to winning more voters than there actually are in the district. For example, you can set x_2 to 200, x_3 to 200, and $x_1 = x_4 = 0$. That solution is feasible, yet it seems to say that you will win 400,000 suburban voters, even though there are only 200,000 actual suburban voters. What constraints can you add to the linear program to ensure that you never seem to win more voters than there actually are? Even if you don't add these constraints, argue that the optimal solution to this linear program can never win more voters than there actually are in the district.

29.2 Formulating problems as linear programs

Linear programming has many applications. Any textbook on operations research is filled with examples of linear programming, and linear programming has become a standard tool taught to students in most business schools. The election scenario is one typical example. Here are two more examples:

- An airline wishes to schedule its flight crews. The Federal Aviation Administration imposes several constraints, such as limiting the number of consecutive hours that each crew member can work and insisting that a particular crew work only on one model of aircraft during each month. The airline wants to schedule crews on all of its flights using as few crew members as possible.
- An oil company wants to decide where to drill for oil. Siting a drill at a particular location has an associated cost and, based on geological surveys, an expected payoff of some number of barrels of oil. The company has a limited budget for locating new drills and wants to maximize the amount of oil it expects to find, given this budget.

Linear programs also model and solve graph and combinatorial problems, such as those appearing in this book. We have already seen a special case of linear programming used to solve systems of difference constraints in Section 22.4. In this section, we'll study how to formulate several graph and network-flow problems as linear programs. Section 35.4 uses linear programming as a tool to find an approximate solution to another graph problem.

Perhaps the most important aspect of linear programming is to be able to recognize when you can formulate a problem as a linear program. Once you cast a problem as a polynomial-sized linear program, you can solve it in polynomial time by the ellipsoid algorithm or interior-point methods. Several linear-programming software packages can solve problems efficiently, so that once the problem is in the form of a linear program, such a package can solve it.

We'll look at several concrete examples of linear-programming problems. We start with two problems that we have already studied: the single-source shortest-paths problem from Chapter 22 and the maximum-flow problem from Chapter 24. We then describe the minimum-cost-flow problem. (Although the minimum-cost-flow problem has a polynomial-time algorithm that is not based on linear programming, we won't describe the algorithm.) Finally, we describe the multicommodity-flow problem, for which the only known polynomial-time algorithm is based on linear programming.

When we solved graph problems in Part VI, we used attribute notation, such as $v.d$ and $(u, v).f$. Linear programs typically use subscripted variables rather than

objects with attached attributes, however. Therefore, when we express variables in linear programs, we indicate vertices and edges through subscripts. For example, we denote the shortest-path weight for vertex v not by $v.d$ but by d_v, and we denote the flow from vertex u to vertex v not by $(u, v).f$ but by f_{uv}. For quantities that are given as inputs to problems, such as edge weights or capacities, we continue to use notations such as $w(u, v)$ and $c(u, v)$.

Shortest paths

We can formulate the single-source shortest-paths problem as a linear program. We'll focus on how to formulate the single-pair shortest-path problem, leaving the extension to the more general single-source shortest-paths problem as Exercise 29.2-2.

In the single-pair shortest-path problem, the input is a weighted, directed graph $G = (V, E)$, with weight function $w : E \to \mathbb{R}$ mapping edges to real-valued weights, a source vertex s, and destination vertex t. The goal is to compute the value d_t, which is the weight of a shortest path from s to t. To express this problem as a linear program, you need to determine a set of variables and constraints that define when you have a shortest path from s to t. The triangle inequality (Lemma 22.10 on page 633) gives $d_v \le d_u + w(u, v)$ for each edge $(u, v) \in E$. The source vertex initially receives a value $d_s = 0$, which never changes. Thus the following linear program expresses the shortest-path weight from s to t:

$$\text{maximize} \quad d_t \tag{29.22}$$

$$\text{subject to}$$

$$d_v \le d_u + w(u, v) \quad \text{for each edge } (u, v) \in E \tag{29.23}$$

$$d_s = 0 \ . \tag{29.24}$$

You might be surprised that this linear program maximizes an objective function when it is supposed to compute shortest paths. Minimizing the objective function would be a mistake, because when all the edge weights are nonnegative, setting $\bar{d}_v = 0$ for all $v \in V$ (recall that a bar over a variable name denotes a specific setting of the variable's value) would yield an optimal solution to the linear program without solving the shortest-paths problem. Maximizing is the right thing to do because an optimal solution to the shortest-paths problem sets each $\bar{d}_v$ to $\min\{\bar{d}_u + w(u, v) : u \in V \text{ and } (u, v) \in E\}$, so that $\bar{d}_v$ is the largest value that is less than or equal to all of the values in the set $\{\bar{d}_u + w(u, v)\}$. Therefore, it makes sense to maximize d_v for all vertices v on a shortest path from s to t subject to these constraints, and maximizing d_t achieves this goal.

This linear program has $|V|$ variables d_v, one for each vertex $v \in V$. It also has $|E| + 1$ constraints: one for each edge, plus the additional constraint that the source vertex's shortest-path weight always has the value 0.

Maximum flow

Next, let's express the maximum-flow problem as a linear program. Recall that the input is a directed graph $G = (V, E)$ in which each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$, and two distinguished vertices: a source s and a sink t. As defined in Section 24.1, a flow is a nonnegative real-valued function $f : V \times V \to \mathbb{R}$ that satisfies the capacity constraint and flow conservation. A maximum flow is a flow that satisfies these constraints and maximizes the flow value, which is the total flow coming out of the source minus the total flow into the source. A flow, therefore, satisfies linear constraints, and the value of a flow is a linear function. Recalling also that we assume that $c(u, v) = 0$ if $(u, v) \notin E$ and that there are no antiparallel edges, the maximum-flow problem can be expressed as a linear program:

$$\text{maximize} \quad \sum_{v \in V} f_{sv} - \sum_{v \in V} f_{vs} \tag{29.25}$$

subject to

$$f_{uv} \leq c(u, v) \quad \text{for each } u, v \in V \tag{29.26}$$

$$\sum_{v \in V} f_{vu} = \sum_{v \in V} f_{uv} \quad \text{for each } u \in V - \{s, t\} \tag{29.27}$$

$$f_{uv} \geq 0 \qquad \text{for each } u, v \in V \ . \tag{29.28}$$

This linear program has $|V|^2$ variables, corresponding to the flow between each pair of vertices, and it has $2|V|^2 + |V| - 2$ constraints.

It is usually more efficient to solve a smaller-sized linear program. The linear program in (29.25)–(29.28) has, for ease of notation, a flow and capacity of 0 for each pair of vertices u, v with $(u, v) \notin E$. It is more efficient to rewrite the linear program so that it has $O(V + E)$ constraints. Exercise 29.2-4 asks you to do so.

Minimum-cost flow

In this section, we have used linear programming to solve problems for which we already knew efficient algorithms. In fact, an efficient algorithm designed specifically for a problem, such as Dijkstra's algorithm for the single-source shortest-paths problem, will often be more efficient than linear programming, both in theory and in practice.

The real power of linear programming comes from the ability to solve new problems. Recall the problem faced by the politician in the beginning of this chapter. The problem of obtaining a sufficient number of votes, while not spending too much money, is not solved by any of the algorithms that we have studied in this book, yet it can be solved by linear programming. Books abound with such real-world problems that linear programming can solve. Linear programming is also

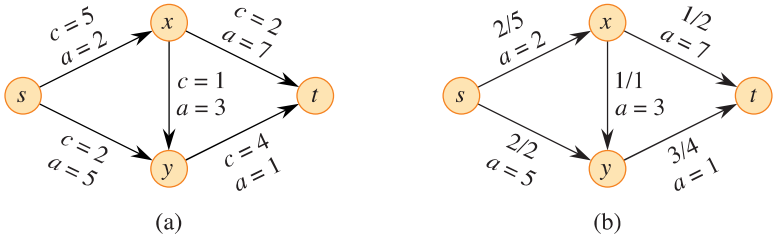

Figure 29.3 **(a)** An example of a minimum-cost-flow problem. Capacities are denoted by c and costs by a. Vertex s is the source, and vertex t is the sink. The goal is to send 4 units of flow from s to t. **(b)** A solution to the minimum-cost flow problem in which 4 units of flow are sent from s to t. For each edge, the flow and capacity are written as flow/capacity.

particularly useful for solving variants of problems for which we may not already know of an efficient algorithm.

Consider, for example, the following generalization of the maximum-flow problem. Suppose that, in addition to a capacity $c(u, v)$ for each edge (u, v), you are given a real-valued cost $a(u, v)$. As in the maximum-flow problem, assume that $c(u, v) = 0$ if $(u, v) \notin E$ and that there are no antiparallel edges. If you send f_{uv} units of flow over edge (u, v), you incur a cost of $a(u, v) \cdot f_{uv}$. You are also given a flow demand d. You wish to send d units of flow from s to t while minimizing the total cost $\sum_{(u,v)\in E} a(u, v) \cdot f_{uv}$ incurred by the flow. This problem is known as the ***minimum-cost-flow problem***.

Figure 29.3(a) shows an example of the minimum-cost-flow problem, with a goal of sending 4 units of flow from s to t while incurring the minimum total cost. Any particular legal flow, that is, a function f satisfying constraints (29.26)–(29.28), incurs a total cost of $\sum_{(u,v)\in E} a(u, v) \cdot f_{uv}$. What is the particular 4-unit flow that minimizes this cost? Figure 29.3(b) shows an optimal solution, with total cost $\sum_{(u,v)\in E} a(u, v) \cdot f_{uv} = (2 \cdot 2) + (5 \cdot 2) + (3 \cdot 1) + (7 \cdot 1) + (1 \cdot 3) = 27$.

There are polynomial-time algorithms specifically designed for the minimum-cost-flow problem, but they are beyond the scope of this book. The minimum-cost-flow problem can be expressed as a linear program, however. The linear program looks similar to the one for the maximum-flow problem with the additional constraint that the value of the flow must be exactly d units, and with the new objective function of minimizing the cost:

$$\text{minimize} \quad \sum_{(u,v)\in E} a(u,v) \cdot f_{uv} \tag{29.29}$$

subject to

$$\begin{aligned}
f_{uv} &\le c(u,v) && \text{for each } u, v \in V \\
\sum_{v\in V} f_{vu} - \sum_{v\in V} f_{uv} &= 0 && \text{for each } u \in V - \{s,t\} \\
\sum_{v\in V} f_{sv} - \sum_{v\in V} f_{vs} &= d \\
f_{uv} &\ge 0 && \text{for each } u, v \in V \ .
\end{aligned} \tag{29.30}$$

Multicommodity flow

As a final example, let's consider another flow problem. Suppose that the Lucky Puck company from Section 24.1 decides to diversify its product line and ship not only hockey pucks, but also hockey sticks and hockey helmets. Each piece of equipment is manufactured in its own factory, has its own warehouse, and must be shipped, each day, from factory to warehouse. The sticks are manufactured in Vancouver and are needed in Saskatoon, and the helmets are manufactured in Edmonton and must be shipped to Regina. The capacity of the shipping network does not change, however, and the different items, or ***commodities***, must share the same network.

This example is an instance of a ***multicommodity-flow problem***. The input to this problem is once again a directed graph $G = (V, E)$ in which each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \ge 0$. As in the maximum-flow problem, implicitly assume that $c(u, v) = 0$ for $(u, v) \notin E$ and that the graph has no antiparallel edges. In addition, there are k different commodities, $K_1, K_2, \ldots, K_k$, with commodity i specified by the triple $K_i = (s_i, t_i, d_i)$. Here, vertex s_i is the source of commodity i, vertex t_i is the sink of commodity i, and d_i is the demand for commodity i, which is the desired flow value for the commodity from s_i to t_i. We define a flow for commodity i, denoted by f_i, (so that f_{iuv} is the flow of commodity i from vertex u to vertex v) to be a real-valued function that satisfies the flow-conservation and capacity constraints. We define f_{uv}, the ***aggregate flow***, to be the sum of the various commodity flows, so that $f_{uv} = \sum_{i=1}^{k} f_{iuv}$. The aggregate flow on edge (u, v) must be no more than the capacity of edge (u, v). This problem has no objective function: the question is to determine whether such a flow exists. Thus the linear program for this problem has a "null" objective function:

$$
\begin{aligned}
&\text{minimize} && 0 && && \\
&\text{subject to} && && && \\
& && \sum_{i=1}^{k} f_{iuv} &\le c(u,v) && \text{for each } u, v \in V \\
& \sum_{v \in V} f_{iuv} - \sum_{v \in V} f_{ivu} &= 0 && && \text{for each } i = 1, 2, \ldots, k \text{ and} \\
& && && && \text{for each } u \in V - \{s_i, t_i\} \\
& \sum_{v \in V} f_{i,s_i,v} - \sum_{v \in V} f_{i,v,s_i} &= d_i && && \text{for each } i = 1, 2, \ldots, k \\
& && f_{iuv} &\ge 0 && \text{for each } u, v \in V \text{ and} \\
& && && && \text{for each } i = 1, 2, \ldots, k \ .
\end{aligned}
$$

The only known polynomial-time algorithm for this problem expresses it as a linear program and then solves it with a polynomial-time linear-programming algorithm.

Exercises

29.2-1

Write out explicitly the linear program corresponding to finding the shortest path from vertex s to vertex x in Figure 22.2(a) on page 609.

29.2-2

Given a graph G, write a linear program for the single-source shortest-paths problem. The solution should have the property that d_v is the shortest-path weight from the source vertex s to v for each vertex $v \in V$.

29.2-3

Write out explicitly the linear program corresponding to finding the maximum flow in Figure 24.1(a).

29.2-4

Rewrite the linear program for maximum flow (29.25)–(29.28) so that it uses only $O(V + E)$ constraints.

29.2-5

Write a linear program that, given a bipartite graph $G = (V, E)$, solves the maximum-bipartite-matching problem.

29.2-6

There can be more than one way to model a particular problem as a linear program. This exercise gives an alternative formulation for the maximum-flow problem. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_p\}$ be the set of *all* possible directed simple paths from source s

to sink t. Using decision variables $x_1, \ldots, x_p$, where x_i is the amount of flow on path i, formulate a linear program for the maximum-flow problem. What is an upper bound on p, the number of directed simple paths from s to t?

29.2-7

In the ***minimum-cost multicommodity-flow problem***, the input is a directed graph $G = (V, E)$ in which each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$ and a cost $a(u, v)$. As in the multicommodity-flow problem, there are k different commodities, $K_1, K_2, \ldots, K_k$, with commodity i specified by the triple $K_i = (s_i, t_i, d_i)$. We define the flow f_i for commodity i and the aggregate flow f_{uv} on edge (u, v) as in the multicommodity-flow problem. A feasible flow is one in which the aggregate flow on each edge (u, v) is no more than the capacity of edge (u, v). The cost of a flow is $\sum_{u,v \in V} a(u, v) \cdot f_{uv}$, and the goal is to find the feasible flow of minimum cost. Express this problem as a linear program.

29.3 Duality

We will now introduce a powerful concept called ***linear-programming duality***. In general, given a maximization problem, duality allows you to formulate a related minimization problem that has the same objective value. The idea of duality is actually more general than linear programming, but we restrict our attention to linear programming in this section.

Duality enables us to prove that a solution is indeed optimal. We saw an example of duality in Chapter 24 with Theorem 24.6, the max-flow min-cut theorem. Suppose that, given an instance of a maximum-flow problem, you find a flow f with value $|f|$. How do you know whether f is a maximum flow? By the max-flow min-cut theorem, if you can find a cut whose value is also $|f|$, then you have verified that f is indeed a maximum flow. This relationship provides an example of duality: given a maximization problem, define a related minimization problem such that the two problems have the same optimal objective values.

Given a linear program in standard form in which the objective is to maximize, let's see how to formulate a ***dual*** linear program in which the objective is to minimize and whose optimal value is identical to that of the original linear program. When referring to dual linear programs, we call the original linear program the ***primal***.

Given the primal linear program

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j \tag{29.31}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i \quad \text{for } i = 1, 2, \ldots, m \tag{29.32}$$

$$x_j \ge 0 \quad \text{for } j = 1, 2, \ldots, n \ , \tag{29.33}$$

its dual is

$$\text{minimize} \quad \sum_{i=1}^{m} b_i y_i \tag{29.34}$$

subject to

$$\sum_{i=1}^{m} a_{ij} y_i \ge c_j \quad \text{for } j = 1, 2, \ldots, n \tag{29.35}$$

$$y_i \ge 0 \quad \text{for } i = 1, 2, \ldots, m \ . \tag{29.36}$$

Mechanically, to form the dual, change the maximization to a minimization, exchange the roles of coefficients on the right-hand sides and in the objective function, and replace each $\le$ by $\ge$. Each of the m constraints in the primal corresponds to a variable y_i in the dual. Likewise, each of the n constraints in the dual corresponds to a variable x_j in the primal. For example, consider the following primal linear program:

$$\text{maximize} \quad 3x_1 + x_2 + 4x_3 \tag{29.37}$$

subject to

$$x_1 + x_2 + 3x_3 \le 30 \tag{29.38}$$

$$2x_1 + 2x_2 + 5x_3 \le 24 \tag{29.39}$$

$$4x_1 + x_2 + 2x_3 \le 36 \tag{29.40}$$

$$x_1, x_2, x_3 \ge 0 \ . \tag{29.41}$$

Its dual is

$$\text{minimize} \quad 30y_1 + 24y_2 + 36y_3 \tag{29.42}$$

subject to

$$y_1 + 2y_2 + 4y_3 \ge 3 \tag{29.43}$$

$$y_1 + 2y_2 + y_3 \ge 1 \tag{29.44}$$

$$3y_1 + 5y_2 + 2y_3 \ge 4 \tag{29.45}$$

$$y_1, y_2, y_3 \ge 0 \ . \tag{29.46}$$

Although forming the dual can be considered a mechanical operation, there is an intuitive explanation. Consider the primal maximization problem (29.37)–(29.41). Each constraint gives an upper bound on the objective function. In addition, if you

take one or more constraints and add together nonnegative multiples of them, you get a valid constraint. For example, you can add constraints (29.38) and (29.39) to obtain the constraint $3x_1 + 3x_2 + 8x_3 \leq 54$. Any feasible solution to the primal must satisfy this new constraint, but there is something else interesting about it. Comparing this new constraint to the objective function (29.37), you can see that for each variable, the corresponding coefficient is at least as large as the coefficient in the objective function. Thus, since the variables x_1, x_2 and x_3 are nonnegative, we have that

$$3x_1 + x_2 + 4x_3 \leq 3x_1 + 3x_2 + 8x_3 \leq 54 ,$$

and so the solution value to the primal is at most 54. In other words, adding these two constraints together has generated an upper bound on the objective value.

In general, for any nonnegative multipliers y_1, y_2, and y_3, you can generate a constraint

$$y_1(x_1+x_2+3x_3)+y_2(2x_1+2x_2+5x_3)+y_3(4x_1+x_2+2x_3) \leq 30y_1+24y_2+36y_3$$

from the primal constraints or, by distributing and regrouping,

$$(y_1+2y_2+4y_3)x_1+(y_1+2y_2+y_3)x_2+(3y_1+5y_2+2y_3)x_3 \leq 30y_1+24y_2+36y_3 .$$

Now, as long as this constraint has coefficients of x_1, x_2, and x_3 that are at least their objective-function coefficients, it is a valid upper bound. That is, as long as

$$\begin{aligned} y_1 + 2y_2 + 4y_3 &\geq 3 , \\ y_1 + 2y_2 + y_3 &\geq 1 , \\ 3y_1 + 5y_2 + 2y_3 &\geq 4 , \end{aligned}$$

you have a valid upper bound of $30y_1+24y_2+36y_3$. The multipliers $y_1, y_2,$ and y_3 must be nonnegative, because otherwise you cannot combine the inequalities. Of course, you would like the upper bound to be as small as possible, and so you want to choose y to minimize $30y_1 + 24y_2 + 36y_3$. Observe that we have just described the dual linear program as the problem of finding the smallest possible upper bound on the primal.

We'll formalize this idea and show in Theorem 29.4 that, if the linear program and its dual are feasible and bounded, then the optimal value of the dual linear program is always equal to the optimal value of the primal linear program. We begin by demonstrating ***weak duality***, which states that any feasible solution to the primal linear program has a value no greater than that of any feasible solution to the dual linear program.

Lemma 29.1 (Weak linear-programming duality)
Let $\bar{x}$ be any feasible solution to the primal linear program in (29.31)–(29.33), and let $\bar{y}$ be any feasible solution to its dual linear program in (29.34)–(29.36). Then

$$\sum_{j=1}^{n} c_j \bar{x}_j \leq \sum_{i=1}^{m} b_i \bar{y}_i \ .$$

Proof We have

$$\begin{aligned}
\sum_{j=1}^{n} c_j \bar{x}_j &\leq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} \bar{y}_i \right) \bar{x}_j \quad \text{(by inequalities (29.35))} \\
&= \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} \bar{x}_j \right) \bar{y}_i \\
&\leq \sum_{i=1}^{m} b_i \bar{y}_i \qquad \text{(by inequalities (29.32))} \ .
\end{aligned}$$

■

Corollary 29.2

Let $\bar{x}$ be a feasible solution to the primal linear program in (29.31)–(29.33), and let $\bar{y}$ be a feasible solution to its dual linear program in (29.34)–(29.36). If

$$\sum_{j=1}^{n} c_j \bar{x}_j = \sum_{i=1}^{m} b_i \bar{y}_i \ ,$$

then $\bar{x}$ and $\bar{y}$ are optimal solutions to the primal and dual linear programs, respectively.

Proof By Lemma 29.1, the objective value of a feasible solution to the primal cannot exceed that of a feasible solution to the dual. The primal linear program is a maximization problem and the dual is a minimization problem. Thus, if feasible solutions $\bar{x}$ and $\bar{y}$ have the same objective value, neither can be improved. ■

We now show that, at optimality, the primal and dual objective values are indeed equal. To prove linear programming duality, we will require one lemma from linear algebra, known as Farkas's lemma, the proof of which Problem 29-4 asks you to provide. Farkas's lemma can take several forms, each of which is about when a set of linear equalities has a solution. In stating the lemma, we use $m + 1$ as a dimension because it matches our use below.

Lemma 29.3 (Farkas's lemma)

Given $M \in \mathbb{R}^{(m+1)\times n}$ and $g \in \mathbb{R}^{m+1}$, exactly one of the following statements is true:

1. There exists $v \in \mathbb{R}^n$ such that $Mv \leq g$,
2. There exists $w \in \mathbb{R}^{m+1}$ such that $w \geq 0, w^{\mathrm{T}}M = 0$ (an n-vector of all zeros), and $w^{\mathrm{T}}g < 0$.

■

Theorem 29.4 (Linear-programming duality)
Given the primal linear program in (29.31)–(29.33) and its corresponding dual in (29.34)–(29.36), if both are feasible and bounded, then for optimal solutions x^* and y^*, we have $c^{\mathrm{T}}x^* = b^{\mathrm{T}}y^*$.

Proof Let $\mu = b^{\mathrm{T}}y^*$ be the optimal value of the dual linear program given in (29.34)–(29.36). Consider an augmented set of primal constraints in which we add a constraint to (29.31)–(29.33) that the objective value is at least μ. We write out this ***augmented primal*** as

$$Ax \le b\,, \tag{29.47}$$
$$c^{\mathrm{T}}x \ge \mu\,. \tag{29.48}$$

We can multiply (29.48) through by -1 and rewrite (29.47)–(29.48) as

$$\begin{pmatrix} A \\ -c^{\mathrm{T}} \end{pmatrix} x \le \begin{pmatrix} b \\ -\mu \end{pmatrix}. \tag{29.49}$$

Here, $\begin{pmatrix} A \\ -c^{\mathrm{T}} \end{pmatrix}$ denotes an $(m+1)\times n$ matrix, x is an n-vector, and $\begin{pmatrix} b \\ -\mu \end{pmatrix}$ denotes an $(m+1)$-vector.

We claim that if there is a feasible solution $\bar{x}$ to the augmented primal, then the theorem is proved. To establish this claim, observe that $\bar{x}$ is also a feasible solution to the original primal and that it has objective value at least μ. We can then apply Lemma 29.1, which states that the objective value of the primal is at most μ, to complete the proof of the theorem.

It therefore remains to show that the augmented primal has a feasible solution. Suppose, for the purpose of contradiction, that the augmented primal is infeasible, which means that there is no $v \in \mathbb{R}^n$ such that $\begin{pmatrix} A \\ -c^{\mathrm{T}} \end{pmatrix} v \le \begin{pmatrix} b \\ -\mu \end{pmatrix}$. We can apply Farkas's lemma, Lemma 29.3, to inequalty (29.49) with

$$M = \begin{pmatrix} A \\ -c^{\mathrm{T}} \end{pmatrix} \text{ and } g = \begin{pmatrix} b \\ -\mu \end{pmatrix}.$$

Because the augmented primal is infeasible, condition 1 of Farkas's lemma does not hold. Therefore, condition 2 must apply, so that there must exist a $w \in \mathbb{R}^{m+1}$ such that $w \ge 0$, $w^{\mathrm{T}}M = 0$, and $w^{\mathrm{T}}g < 0$. Let's write w as $w = \begin{pmatrix} \bar{y} \\ \lambda \end{pmatrix}$ for some $\bar{y} \in \mathbb{R}^m$ and $\lambda \in \mathbb{R}$, where $\bar{y} \ge 0$ and $\lambda \ge 0$. Substituting for w, M, and g in condition 2 gives

$$\begin{pmatrix} \bar{y} \\ \lambda \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} A \\ -c^{\mathrm{T}} \end{pmatrix} = 0 \text{ and } \begin{pmatrix} \bar{y} \\ \lambda \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} b \\ -\mu \end{pmatrix} < 0\,.$$

Unpacking the matrix notation gives

$$\bar{y}^{\mathrm{T}}A - \lambda c^{\mathrm{T}} = 0 \text{ and } \bar{y}^{\mathrm{T}}b - \lambda\mu < 0 \,. \tag{29.50}$$

We now show that the requirements in (29.50) contradict the assumption that μ is the optimal solution value for the dual linear program. We consider two cases.

The first case is when $\lambda = 0$. In this case, (29.50) simplifies to

$$\bar{y}^{\mathrm{T}}A = 0 \text{ and } \bar{y}^{\mathrm{T}}b < 0 \,. \tag{29.51}$$

We'll now construct a dual feasible solution y' with an objective value smaller than $b^{\mathrm{T}}y^*$. Set $y' = y^* + \epsilon\bar{y}$, for any $\epsilon > 0$. Since

$$\begin{aligned} y'^{\mathrm{T}}A &= (y^* + \epsilon\bar{y})^{\mathrm{T}}A \\ &= y^{*\mathrm{T}}A + \epsilon\bar{y}^{\mathrm{T}}A \\ &= y^{*\mathrm{T}}A && \text{(by (29.51))} \\ &\geq c^{\mathrm{T}} && \text{(because } y^* \text{ is feasible)} \,, \end{aligned}$$

y' is feasible. Now consider the objective value

$$\begin{aligned} b^{\mathrm{T}}y' &= b^{\mathrm{T}}(y^* + \epsilon\bar{y}) \\ &= b^{\mathrm{T}}y^* + \epsilon b^{\mathrm{T}}\bar{y} \\ &< b^{\mathrm{T}}y^* \,, \end{aligned}$$

where the last inequality follows because $\epsilon > 0$ and, by (29.51), $\bar{y}^{\mathrm{T}}b = b^{\mathrm{T}}\bar{y} < 0$ (since both $\bar{y}^{\mathrm{T}}b$ and $b^{\mathrm{T}}\bar{y}$ are the inner product of b and $\bar{y}$), and so their product is negative. Thus we have a feasible dual solution of value less than μ, which contradicts μ being the optimal objective value.

We now consider the second case, where $\lambda > 0$. In this case, we can take (29.50) and divide through by λ to obtain

$$(\bar{y}^{\mathrm{T}}/\lambda)A - (\lambda/\lambda)c^{\mathrm{T}} = 0 \text{ and } (\bar{y}^{\mathrm{T}}/\lambda)b - (\lambda/\lambda)\mu < 0 \,. \tag{29.52}$$

Now set $y' = \bar{y}/\lambda$ in (29.52), giving

$$y'^{\mathrm{T}}A = c^{\mathrm{T}} \text{ and } y'^{\mathrm{T}}b < \mu \,.$$

Thus, y' is a feasible dual solution with objective value strictly less than μ, a contradiction. We conclude that the augmented primal has a feasible solution, and the theorem is proved. ■

Fundamental theorem of linear programming

We conclude this chapter by stating the fundamental theorem of linear programming, which extends Theorem 29.4 to the cases when the linear program may be either feasible or unbounded. Exercise 29.3-8 asks you to provide the proof.

Theorem 29.5 (Fundamental theorem of linear programming)
Any linear program, given in standard form, either

1. has an optimal solution with a finite objective value,
2. is infeasible, or
3. is unbounded. ■

Exercises

29.3-1
Formulate the dual of the linear program given in lines (29.6)–(29.10) on page 852.

29.3-2
You have a linear program that is not in standard form. You could produce the dual by first converting it to standard form, and then taking the dual. It would be more convenient, however, to produce the dual directly. Explain how to directly take the dual of an arbitrary linear program.

29.3-3
Write down the dual of the maximum-flow linear program, as given in lines (29.25)–(29.28) on page 862. Explain how to interpret this formulation as a minimum-cut problem.

29.3-4
Write down the dual of the minimum-cost-flow linear program, as given in lines (29.29)–(29.30) on page 864. Explain how to interpret this problem in terms of graphs and flows.

29.3-5
Show that the dual of the dual of a linear program is the primal linear program.

29.3-6
Which result from Chapter 24 can be interpreted as weak duality for the maximum-flow problem?

29.3-7
Consider the following 1-variable primal linear program:

$$\begin{aligned} &\text{maximize} && tx \\ &\text{subject to} \\ &&& rx \le s \\ &&& x \ge 0 \ , \end{aligned}$$

where r, s, and t are arbitrary real numbers. State for which values of r, s, and t you can assert that

1. Both the primal linear program and its dual have optimal solutions with finite objective values.
2. The primal is feasible, but the dual is infeasible.
3. The dual is feasible, but the primal is infeasible.
4. Neither the primal nor the dual is feasible.

29.3-8
Prove the fundamental theorem of linear programming, Theorem 29.5.

Problems

29-1 Linear-inequality feasibility
Given a set of m linear inequalities on n variables $x_1, x_2, \ldots, x_n$, the ***linear-inequality feasibility problem*** asks whether there is a setting of the variables that simultaneously satisfies each of the inequalities.

a. Given an algorithm for the linear-programming problem, show how to use it to solve a linear-inequality feasibility problem. The number of variables and constraints that you use in the linear-programming problem should be polynomial in n and m.

b. Given an algorithm for the linear-inequality feasibility problem, show how to use it to solve a linear-programming problem. The number of variables and linear inequalities that you use in the linear-inequality feasibility problem should be polynomial in n and m, the number of variables and constraints in the linear program.

29-2 Complementary slackness
Complementary slackness describes a relationship between the values of primal variables and dual constraints and between the values of dual variables and primal constraints. Let $\bar{x}$ be a feasible solution to the primal linear program given in (29.31)–(29.33), and let $\bar{y}$ be a feasible solution to the dual linear program given in (29.34)–(29.36). Complementary slackness states that the following conditions are necessary and sufficient for $\bar{x}$ and $\bar{y}$ to be optimal:

$$\sum_{i=1}^{m} a_{ij}\bar{y}_i = c_j \quad \text{or} \quad \bar{x}_j = 0 \quad \text{for } j = 1, 2, \ldots, n$$

and

$$\sum_{j=1}^{n} a_{ij}\bar{x}_j = b_i \quad \text{or} \quad \bar{y}_i = 0 \quad \text{for } i = 1, 2, \ldots, m\,.$$

a. Verify that complementary slackness holds for the linear program in lines (29.37)–(29.41).

b. Prove that complementary slackness holds for any primal linear program and its corresponding dual.

c. Prove that a feasible solution $\bar{x}$ to a primal linear program given in lines (29.31)–(29.33) is optimal if and only if there exist values $\bar{y} = (\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_m)$ such that

1. $\bar{y}$ is a feasible solution to the dual linear program given in (29.34)–(29.36),
2. $\sum_{i=1}^{m} a_{ij}\bar{y}_i = c_j$ for all j such that $\bar{x}_j > 0$, and
3. $\bar{y}_i = 0$ for all i such that $\sum_{j=1}^{n} a_{ij}\bar{x}_j < b_i$.

29-3 Integer linear programming

An ***integer linear-programming problem*** is a linear-programming problem with the additional constraint that the variables x must take on integer values. Exercise 34.5-3 on page 1098 shows that just determining whether an integer linear program has a feasible solution is NP-hard, which means that there is no known polynomial-time algorithm for this problem.

a. Show that weak duality (Lemma 29.1) holds for an integer linear program.

b. Show that duality (Theorem 29.4) does not always hold for an integer linear program.

c. Given a primal linear program in standard form, let P be the optimal objective value for the primal linear program, D be the optimal objective value for its dual, IP be the optimal objective value for the integer version of the primal (that is, the primal with the added constraint that the variables take on integer values), and ID be the optimal objective value for the integer version of the dual. Assuming that both the primal integer program and the dual integer program are feasible and bounded, show that

$$IP \le P = D \le ID\,.$$

29-4 Farkas's lemma

Prove Farkas's lemma, Lemma 29.3.

29-5 Minimum-cost circulation

This problem considers a variant of the minimum-cost-flow problem from Section 29.2 in which there is no demand, source, or sink. Instead, the input, as before, contains a flow network, capacity constraints $c(u, v)$, and edge costs $a(u, v)$. A flow is feasible if it satisfies the capacity constraint on every edge and flow conservation at *every* vertex. The goal is to find, among all feasible flows, the one of minimum cost. We call this problem the ***minimum-cost-circulation problem.***

a. Formulate the minimum-cost-circulation problem as a linear program.

b. Suppose that for all edges $(u, v) \in E$, we have $a(u, v) > 0$. What does an optimal solution to the minimum-cost-circulation problem look like?

c. Formulate the maximum-flow problem as a minimum-cost-circulation problem linear program. That is, given a maximum-flow problem instance $G = (V, E)$ with source s, sink t and edge capacities c, create a minimum-cost-circulation problem by giving a (possibly different) network $G' = (V', E')$ with edge capacities c' and edge costs a' such that you can derive a solution to the maximum-flow problem from a solution to the minimum-cost-circulation problem.

d. Formulate the single-source shortest-path problem as a minimum-cost-circulation problem linear program.

Chapter notes

This chapter only begins to study the wide field of linear programming. A number of books are devoted exclusively to linear programming, including those by Chvátal [94], Gass [178], Karloff [246], Schrijver [398], and Vanderbei [444]. Many other books give a good coverage of linear programming, including those by Papadimitriou and Steiglitz [353] and Ahuja, Magnanti, and Orlin [7]. The coverage in this chapter draws on the approach taken by Chvátal.

The simplex algorithm for linear programming was invented by G. Dantzig in 1947. Shortly after, researchers discovered how to formulate a number of problems in a variety of fields as linear programs and solve them with the simplex algorithm. As a result, applications of linear programming flourished, along with several algorithms. Variants of the simplex algorithm remain the most popular

methods for solving linear-programming problems. This history appears in a number of places, including the notes in [94] and [246].

The ellipsoid algorithm was the first polynomial-time algorithm for linear programming and is due to L. G. Khachian in 1979. It was based on earlier work by N. Z. Shor, D. B. Judin, and A. S. Nemirovskii. Grötschel, Lovász, and Schrijver [201] describe how to use the ellipsoid algorithm to solve a variety of problems in combinatorial optimization. To date, the ellipsoid algorithm does not appear to be competitive with the simplex algorithm in practice.

Karmarkar's paper [247] includes a description of the first interior-point algorithm. Many subsequent researchers designed interior-point algorithms. Good surveys appear in the article of Goldfarb and Todd [189] and the book by Ye [463].

Analysis of the simplex algorithm remains an active area of research. V. Klee and G. J. Minty constructed an example on which the simplex algorithm runs through $2^n - 1$ iterations. The simplex algorithm usually performs well in practice, and many researchers have tried to give theoretical justification for this empirical observation. A line of research begun by K. H. Borgwardt, and carried on by many others, shows that under certain probabilistic assumptions on the input, the simplex algorithm converges in expected polynomial time. Spielman and Teng [421] made progress in this area, introducing the "smoothed analysis of algorithms" and applying it to the simplex algorithm.

The simplex algorithm is known to run efficiently in certain special cases. Particularly noteworthy is the network-simplex algorithm, which is the simplex algorithm, specialized to network-flow problems. For certain network problems, including the shortest-paths, maximum-flow, and minimum-cost-flow problems, variants of the network-simplex algorithm run in polynomial time. See, for example, the article by Orlin [349] and the citations therein.