29 Linear Programming

Many problems take the form of maximizing or minimizing an objective, given limited resources and competing constraints. If we can specify the objective as a linear function of certain variables, and if we can specify the constraints on resources as equalities or inequalities on those variables, then we 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 in which you do not believe. You realize, however, that certain issues may be more effective in winning votes in certain places. Your primary issues are building more roads, gun control, farm subsidies, and a gasoline tax dedicated to improved public transit. 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 building roads, $0 to gun control, $4,000 to farm subsidies, and $9,000 to a gasoline tax. In this case, you

policy	urban	suburban	rural
build roads	−2	5	3
gun control	8	2	−5
farm subsidies	0	0	10
gasoline tax	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.

would win $20(-2)+0(8)+4(0)+9(10) = 50$ thousand urban votes, $20(5)+0(2)+4(0)+9(0) = 100$ thousand suburban votes, and $20(3)+0(-5)+4(10)+9(-2) = 82$ 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, 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.

Naturally, you may wonder whether this strategy is the best possible. That is, could you achieve your goals while spending less on advertising? Additional trial and error might help you to answer this question, but wouldn't you rather have a systematic method for answering such questions? In order to develop one, we shall formulate this question mathematically. We introduce 4 variables:

- x_1 is the number of thousands of dollars spent on advertising on building roads,
- x_2 is the number of thousands of dollars spent on advertising on gun control,
- x_3 is the number of thousands of dollars spent on advertising on farm subsidies, and
- x_4 is the number of thousands of dollars spent on advertising on a gasoline tax.

We can write the requirement that we win at least 50,000 urban votes as

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

Similarly, we can write the requirements that we 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. 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, we require that

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

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

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

The solution of this linear program yields your optimal strategy.

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

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

and

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

are ***linear inequalities***. We use the general term ***linear constraints*** to denote either linear equalities or linear inequalities. In linear programming, we do 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 we are to minimize, then we call the linear program a ***minimization linear program***, and if we are to maximize, then we call the linear program a ***maximization linear program***.

The remainder of this chapter covers how to formulate and solve linear programs. Although several polynomial-time algorithms for linear programming have been developed, we will not study them in this chapter. Instead, we shall study the simplex algorithm, which is the oldest linear-programming algorithm. The simplex algorithm does not run in polynomial time in the worst case, but it is fairly efficient and widely used in practice.

An overview of linear programming

In order to describe properties of and algorithms for linear programs, we find it convenient to express them in canonical forms. We shall use two forms, ***standard*** and ***slack***, in this chapter. We will define them precisely in Section 29.1. Informally, a linear program in standard form is the maximization of a linear function subject to linear *inequalities*, whereas a linear program in slack form is the maximization of a linear function subject to linear *equalities*. We shall typically use standard form for expressing linear programs, but we find it more convenient to use slack form when we describe the details of the simplex algorithm. For now, we restrict our attention to maximizing a linear function on n variables subject to a set of m linear inequalities.

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

$$\begin{aligned}
&\text{maximize} && x_1 + x_2 && && && && (29.11)\\
&\text{subject to} && && && && && \\
& && 4x_1 - x_2 && \le && 8 && (29.12)\\
& && 2x_1 + x_2 && \le && 10 && (29.13)\\
& && 5x_1 - 2x_2 && \ge && -2 && (29.14)\\
& && x_1, x_2 && \ge && 0\ . && (29.15)
\end{aligned}$$

We call any setting of the variables x_1 and x_2 that satisfies all the constraints (29.12)–(29.15) a ***feasible solution*** to the linear program. If we graph the constraints in the (x_1, x_2)-Cartesian coordinate system, as in Figure 29.2(a), we see

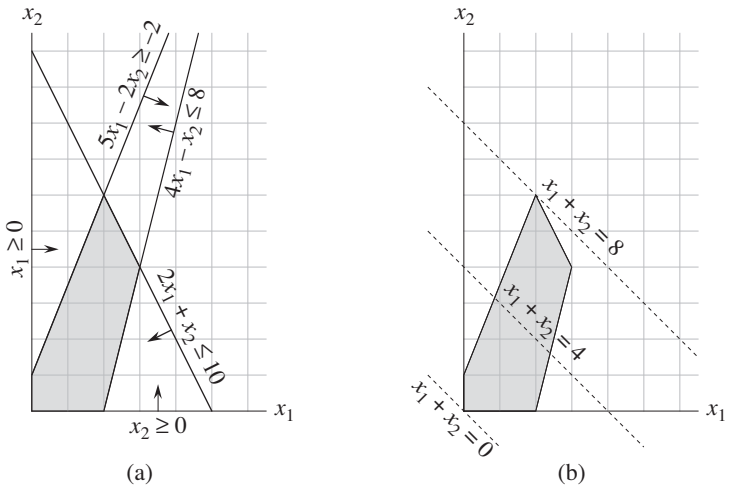


Figure 29.2 **(a)** The linear program given in (29.12)–(29.15). Each constraint is represented by a line and a direction. The intersection of the constraints, which is the feasible region, is shaded. **(b)** The dotted lines show, respectively, the points for which the objective value is 0, 4, and 8. The optimal solution to the linear program is $x_1 = 2$ and $x_2 = 6$ with objective value 8.

that the set of feasible solutions (shaded in the figure) forms a convex region[1] in the two-dimensional space. We call this convex region the ***feasible region*** and the function we wish to maximize the ***objective function***. Conceptually, we could evaluate the objective function $x_1 + x_2$ at each point in the feasible region; we call the value of the objective function at a particular point the ***objective value***. We could then identify a point that has the maximum objective value as an optimal solution. For this example (and for most linear programs), the feasible region contains an infinite number of points, and so we need to determine 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, we 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. If we plot $x_1 + x_2 = 0$, we obtain 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 z, the intersection

[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.

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 at which this occurs is an optimal solution to the linear program, which in this case is the point $x_1 = 2$ 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 we cannot easily graph linear programs with more than two variables, the same intuition holds. If we 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 we move this plane away from the origin, in a direction normal to the objective function, we 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 we 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.

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 29.4,

we shall use a concept called "duality" to show that the solution returned by the simplex algorithm is indeed optimal.

Although the geometric view gives a good intuitive view of the operations of the simplex algorithm, we shall not refer to it explicitly when developing the details of the simplex algorithm in Section 29.3. Instead, we take an algebraic view. We first write the given linear program in slack form, which is a set of linear equalities. These linear equalities express some of the variables, called "basic variables," in terms of other variables, called "nonbasic variables." We move from one vertex to another by making a basic variable become nonbasic and making a nonbasic variable become basic. We call this operation a "pivot" and, viewed algebraically, it is nothing more than rewriting the linear program in an equivalent slack form.

The two-variable example described above was particularly simple. We shall need to address several more details in this chapter. These issues include identifying linear programs that have no solutions, linear programs that have no finite optimal solution, and linear programs for which the origin is not a feasible solution.

Applications of linear programming

Linear programming has a large number of 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. Two more examples of linear programming are the following:

- An airline wishes to schedule its flight crews. The Federal Aviation Administration imposes many 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.

With linear programs, we also model and solve graph and combinatorial problems, such as those appearing in this textbook. We have already seen a special case of linear programming used to solve systems of difference constraints in Section 24.4. In Section 29.2, we shall study how to formulate several graph and network-flow problems as linear programs. In Section 35.4, we shall use linear programming as a tool to find an approximate solution to another graph problem.

Algorithms for linear programming

This chapter studies the simplex algorithm. This 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 we add to a linear program the additional requirement that all variables take on integer values, we have an ***integer linear program***. Exercise 34.5-3 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 problems, there is no known polynomial-time algorithm for integer linear programming. In contrast, we can solve a general linear-programming problem in polynomial time.

In this chapter, if we have a linear program with variables $x = (x_1, x_2, \ldots, x_n)$ and wish to refer to a particular setting of the variables, we shall use the notation $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$.

29.1 Standard and slack forms

This section describes two formats, standard form and slack form, that are useful when we specify and work with linear programs. In standard form, all the constraints are inequalities, whereas in slack form, all constraints are equalities (except for those that require the variables to be nonnegative).

Standard form

In ***standard form***, we are given 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 $j = 1, 2, \ldots, n$. We wish to find n real numbers $x_1, x_2, \ldots, x_n$ that

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j \tag{29.16}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \text{for } i = 1, 2, \ldots, m \tag{29.17}$$

$$x_j \geq 0 \quad \text{for } j = 1, 2, \ldots, n\,. \tag{29.18}$$

Generalizing the terminology we introduced for the two-variable linear program, we call expression (29.16) the ***objective function*** and the $n + m$ inequalities in lines (29.17) and (29.18) the ***constraints***. The n constraints in line (29.18) are the ***nonnegativity constraints***. An arbitrary linear program need not have nonnegativity constraints, but standard form requires them. Sometimes we find it 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 (29.16)–(29.18) as

$$\text{maximize} \quad c^{\mathrm{T}} x \tag{29.19}$$

subject to

$$Ax \leq b \tag{29.20}$$

$$x \geq 0\,. \tag{29.21}$$

In line (29.19), $c^{\mathrm{T}} x$ is the inner product of two vectors. In inequality (29.20), Ax is a matrix-vector product, and in inequality (29.21), $x \geq 0$ means that each entry of the vector x must be nonnegative. We see that we can specify a linear program in standard form by a tuple (A, b, c), and we shall adopt the convention that A, b, and c always have the dimensions given above.

We now introduce terminology to describe solutions to linear programs. We used some of this terminology in the earlier example of a two-variable linear program. We call a setting of the variables $\bar{x}$ that satisfies all the constraints a ***feasible solution***, whereas a setting of the variables $\bar{x}$ that fails to satisfy at least one constraint 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***; otherwise it is ***feasible***. If a linear program has some feasible solutions but does not have a finite optimal objective value, we say that the linear program is ***unbounded***. Exercise 29.1-9 asks you to show that a linear program can have a finite optimal objective value even if the feasible region is not bounded.

Converting linear programs into standard form

It is always possible to convert a linear program, given as minimizing or maximizing a linear function subject to linear constraints, into standard form. A linear program might not be in standard form for any of four possible reasons:

1. The objective function might be a minimization rather than a maximization.
2. There might be variables without nonnegativity constraints.
3. There might be ***equality constraints***, which have an equal sign rather than a less-than-or-equal-to sign.
4. There might be ***inequality constraints***, but instead of having a less-than-or-equal-to sign, they have a greater-than-or-equal-to sign.

When converting one linear program L into another linear program L', we would like the property that an optimal solution to L' yields an optimal solution to L. To capture this idea, we say that two maximization linear programs L and L' are ***equivalent*** if for each feasible solution $\bar{x}$ to L with objective value z, there is a corresponding feasible solution $\bar{x}'$ to L' with objective value z, and for each feasible solution $\bar{x}'$ to L' with objective value z, there is a corresponding feasible solution $\bar{x}$ to L with objective value z. (This definition does not imply a one-to-one correspondence between feasible solutions.) A minimization linear program L and a maximization linear program L' are equivalent if for each feasible solution $\bar{x}$ to L with objective value z, there is a corresponding feasible solution $\bar{x}'$ to L' with objective value $-z$, and for each feasible solution $\bar{x}'$ to L' with objective value z, there is a corresponding feasible solution $\bar{x}$ to L with objective value $-z$.

We now show how to remove, one by one, each of the possible problems in the list above. After removing each one, we shall argue that the new linear program is equivalent to the old one.

To convert a minimization linear program L into an equivalent maximization linear program L', we simply negate the coefficients in the objective function. Since L and L' have identical sets of feasible solutions and, for any feasible solution, the objective value in L is the negative of the objective value in L', these two linear programs are equivalent. For example, if we have the linear program

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

and we negate the coefficients of the objective function, we obtain

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

Next, we show how to convert a linear program in which some of the variables do not have nonnegativity constraints into one in which each variable has a nonnegativity constraint. Suppose that some variable x_j does not have a nonnegativity constraint. Then, we replace each occurrence of x_j by $x'_j - x''_j$, and add the nonnegativity constraints $x'_j \ge 0$ and $x''_j \ge 0$. Thus, if the objective function has a term $c_j x_j$, we replace it by $c_j x'_j - c_j x''_j$, and if constraint i has a term $a_{ij} x_j$, we replace it by $a_{ij} x'_j - a_{ij} x''_j$. Any feasible solution $\hat{x}$ to the new linear program corresponds to a feasible solution $\bar{x}$ to the original linear program with $\bar{x}_j = \hat{x}'_j - \hat{x}''_j$ and with the same objective value. Also, any feasible solution $\bar{x}$ to the original linear program corresponds to a feasible solution $\hat{x}$ to the new linear program with $\hat{x}'_j = \bar{x}_j$ and $\hat{x}''_j = 0$ if $\bar{x}_j \ge 0$, or with $\hat{x}''_j = \bar{x}_j$ and $\hat{x}'_j = 0$ if $\bar{x}_j < 0$. The two linear programs have the same objective value regardless of the sign of $\bar{x}_j$. Thus, the two linear programs are equivalent. We apply this conversion scheme to each variable that does not have a nonnegativity constraint to yield an equivalent linear program in which all variables have nonnegativity constraints.

Continuing the example, we want to ensure that each variable has a corresponding nonnegativity constraint. Variable x_1 has such a constraint, but variable x_2 does not. Therefore, we replace x_2 by two variables x'_2 and x''_2, and we modify the linear program to obtain

$$
\begin{aligned}
\text{maximize} \quad & 2x_1 - 3x'_2 + 3x''_2 \\
\text{subject to} \quad & \\
& x_1 + x'_2 - x''_2 = 7 \\
& x_1 - 2x'_2 + 2x''_2 \le 4 \\
& x_1, x'_2, x''_2 \ge 0 \ .
\end{aligned}
\tag{29.22}
$$

Next, we convert equality constraints into inequality constraints. Suppose that a linear program has an equality constraint $f(x_1, x_2, \ldots, x_n) = b$. Since $x = y$ if and only if both $x \ge y$ and $x \le y$, we can replace this equality constraint by the pair of inequality constraints $f(x_1, x_2, \ldots, x_n) \le b$ and $f(x_1, x_2, \ldots, x_n) \ge b$. Repeating this conversion for each equality constraint yields a linear program in which all constraints are inequalities.

Finally, we can convert the greater-than-or-equal-to constraints to less-than-or-equal-to constraints by multiplying these constraints through by -1. That is, any inequality of the form

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i$$

is equivalent to

$$\sum_{j=1}^{n} -a_{ij} x_j \leq -b_i \ .$$

Thus, by replacing each coefficient a_{ij} by $-a_{ij}$ and each value b_i by $-b_i$, we obtain an equivalent less-than-or-equal-to constraint.

Finishing our example, we replace the equality in constraint (29.22) by two inequalities, obtaining

$$\begin{aligned}
&\text{maximize} && 2x_1 - 3x_2' + 3x_2'' \\
&\text{subject to} \\
&&& x_1 + x_2' - x_2'' \leq 7 \\
&&& x_1 + x_2' - x_2'' \geq 7 \\
&&& x_1 - 2x_2' + 2x_2'' \leq 4 \\
&&& x_1, x_2', x_2'' \geq 0 \ .
\end{aligned} \tag{29.23}$$

Finally, we negate constraint (29.23). For consistency in variable names, we rename x_2' to x_2 and x_2'' to x_3, obtaining the standard form

$$\text{maximize} \quad 2x_1 - 3x_2 + 3x_3 \tag{29.24}$$

$$\text{subject to}$$

$$x_1 + x_2 - x_3 \leq 7 \tag{29.25}$$

$$-x_1 - x_2 + x_3 \leq -7 \tag{29.26}$$

$$x_1 - 2x_2 + 2x_3 \leq 4 \tag{29.27}$$

$$x_1, x_2, x_3 \geq 0 \ . \tag{29.28}$$

Converting linear programs into slack form

To efficiently solve a linear program with the simplex algorithm, we prefer to express it in a form in which some of the constraints are equality constraints. More precisely, we shall convert it into a form in which the nonnegativity constraints are the only inequality constraints, and the remaining constraints are equalities. Let

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \tag{29.29}$$

be an inequality constraint. We introduce a new variable s and rewrite inequality (29.29) as the two constraints

$$s = b_i - \sum_{j=1}^{n} a_{ij} x_j \,, \tag{29.30}$$

$$s \geq 0 \,. \tag{29.31}$$

We call s a ***slack variable*** because it measures the ***slack***, or difference, between the left-hand and right-hand sides of equation (29.29). (We shall soon see why we find it convenient to write the constraint with only the slack variable on the left-hand side.) Because inequality (29.29) is true if and only if both equation (29.30) and inequality (29.31) are true, we can convert each inequality constraint of a linear program in this way to obtain an equivalent linear program in which the only inequality constraints are the nonnegativity constraints. When converting from standard to slack form, we shall use x_{n+i} (instead of s) to denote the slack variable associated with the ith inequality. The ith constraint is therefore

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j \,, \tag{29.32}$$

along with the nonnegativity constraint $x_{n+i} \geq 0$.

By converting each constraint of a linear program in standard form, we obtain a linear program in a different form. For example, for the linear program described in (29.24)–(29.28), we introduce slack variables x_4, x_5, and x_6, obtaining

$$\text{maximize} \qquad 2x_1 - 3x_2 + 3x_3 \tag{29.33}$$

subject to

$$x_4 = 7 - x_1 - x_2 + x_3 \tag{29.34}$$

$$x_5 = -7 + x_1 + x_2 - x_3 \tag{29.35}$$

$$x_6 = 4 - x_1 + 2x_2 - 2x_3 \tag{29.36}$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \,. \tag{29.37}$$

In this linear program, all the constraints except for the nonnegativity constraints are equalities, and each variable is subject to a nonnegativity constraint. We write each equality constraint with one of the variables on the left-hand side of the equality and all others on the right-hand side. Furthermore, each equation has the same set of variables on the right-hand side, and these variables are also the only ones that appear in the objective function. We call the variables on the left-hand side of the equalities ***basic variables*** and those on the right-hand side ***nonbasic variables***.

For linear programs that satisfy these conditions, we shall sometimes omit the words "maximize" and "subject to," as well as the explicit nonnegativity constraints. We shall also use the variable z to denote the value of the objective func-

tion. We call the resulting format ***slack form***. If we write the linear program given in (29.33)–(29.37) in slack form, we obtain

$$\begin{aligned}
z &= 2x_1 - 3x_2 + 3x_3 && (29.38)\\
x_4 &= 7 - x_1 - x_2 + x_3 && (29.39)\\
x_5 &= -7 + x_1 + x_2 - x_3 && (29.40)\\
x_6 &= 4 - x_1 + 2x_2 - 2x_3 \ . && (29.41)
\end{aligned}$$

As with standard form, we find it convenient to have a more concise notation for describing a slack form. As we shall see in Section 29.3, the sets of basic and nonbasic variables will change as the simplex algorithm runs. We use N to denote the set of indices of the nonbasic variables and B to denote the set of indices of the basic variables. We always have that $|N| = n$, $|B| = m$, and $N \cup B = \{1, 2, \ldots, n + m\}$. The equations are indexed by the entries of B, and the variables on the right-hand sides are indexed by the entries of N. As in standard form, we use b_i, c_j, and a_{ij} to denote constant terms and coefficients. We also use v to denote an optional constant term in the objective function. (We shall see a little later that including the constant term in the objective function makes it easy to determine the value of the objective function.) Thus we can concisely define a slack form by a tuple (N, B, A, b, c, v), denoting the slack form

$$\begin{aligned}
z &= v + \sum_{j \in N} c_j x_j && (29.42)\\
x_i &= b_i - \sum_{j \in N} a_{ij} x_j \quad \text{for } i \in B \ , && (29.43)
\end{aligned}$$

in which all variables x are constrained to be nonnegative. Because we subtract the sum $\sum_{j \in N} a_{ij} x_j$ in (29.43), the values a_{ij} are actually the negatives of the coefficients as they "appear" in the slack form.

For example, in the slack form

$$\begin{aligned}
z &= 28 - \frac{x_3}{6} - \frac{x_5}{6} - \frac{2x_6}{3}\\
x_1 &= 8 + \frac{x_3}{6} + \frac{x_5}{6} - \frac{x_6}{3}\\
x_2 &= 4 - \frac{8x_3}{3} - \frac{2x_5}{3} + \frac{x_6}{3}\\
x_4 &= 18 - \frac{x_3}{2} + \frac{x_5}{2} \ ,
\end{aligned}$$

we have $B = \{1, 2, 4\}$, $N = \{3, 5, 6\}$,

$$A = \begin{pmatrix} a_{13} & a_{15} & a_{16} \\ a_{23} & a_{25} & a_{26} \\ a_{43} & a_{45} & a_{46} \end{pmatrix} = \begin{pmatrix} -1/6 & -1/6 & 1/3 \\ 8/3 & 2/3 & -1/3 \\ 1/2 & -1/2 & 0 \end{pmatrix},$$

$$b = \begin{pmatrix} b_1 \\ b_2 \\ b_4 \end{pmatrix} = \begin{pmatrix} 8 \\ 4 \\ 18 \end{pmatrix},$$

$c = \begin{pmatrix} c_3 & c_5 & c_6 \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} -1/6 & -1/6 & -2/3 \end{pmatrix}^{\mathrm{T}}$, and $v = 28$. Note that the indices into A, b, and c are not necessarily sets of contiguous integers; they depend on the index sets B and N. As an example of the entries of A being the negatives of the coefficients as they appear in the slack form, observe that the equation for x_1 includes the term $x_3/6$, yet the coefficient a_{13} is actually $-1/6$ rather than $+1/6$.

Exercises

29.1-1

If we express the linear program in (29.24)–(29.28) in the compact notation of (29.19)–(29.21), what are n, m, A, b, and c?

29.1-2

Give three feasible solutions to the linear program in (29.24)–(29.28). What is the objective value of each one?

29.1-3

For the slack form in (29.38)–(29.41), what are N, B, A, b, c, and v?

29.1-4

Convert the following linear program into standard form:

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

29.1-5

Convert the following linear program into slack form:

$$\begin{aligned}
\text{maximize} \quad & 2x_1 - 6x_3 \\
\text{subject to} \quad & \\
& x_1 + x_2 - x_3 \le 7 \\
& 3x_1 - x_2 \ge 8 \\
& -x_1 + 2x_2 + 2x_3 \ge 0 \\
& x_1, x_2, x_3 \ge 0 \; .
\end{aligned}$$

What are the basic and nonbasic variables?

29.1-6

Show that the following linear program is infeasible:

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

29.1-7

Show that the following linear program is unbounded:

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

29.1-8

Suppose that we have a general linear program with n variables and m constraints, and suppose that we convert it into standard form. Give an upper bound on the number of variables and constraints in the resulting linear program.

29.1-9

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

from the algorithm, which in such cases would increment the flow by an integer amount on each iteration.

Hence integrality comes for free in the maximum-flow problem. Unfortunately, this is the exception rather than the rule: as we will see in Chapter 8, it is a very difficult problem to find the optimum solution (or for that matter, *any* solution) of a general linear program, if we also demand that the variables be integers.

7.4 Duality

We have seen that in networks, flows are smaller than cuts, but the maximum flow and minimum cut exactly coincide and each is therefore a certificate of the other's optimality. Remarkable as this phenomenon is, we now generalize it from maximum flow to *any* problem that can be solved by linear programming! It turns out that every linear maximization problem has a *dual* minimization problem, and they relate to each other in much the same way as flows and cuts.

To understand what duality is about, recall our introductory LP with the two types of chocolate:

$$\begin{aligned} \max\ x_1 + 6x_2 & \\ x_1 &\leq 200 \\ x_2 &\leq 300 \\ x_1 + x_2 &\leq 400 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Simplex declares the optimum solution to be $(x_1, x_2) = (100, 300)$, with objective value 1900. Can this answer be checked somehow? Let's see: suppose we take the first inequality and add it to six times the second inequality. We get

$$x_1 + 6x_2 \ \leq\ 2000.$$

This is interesting, because it tells us that it is impossible to achieve a profit of more than 2000. Can we add together some other combination of the LP constraints and bring this upper bound even closer to 1900? After a little experimentation, we find that multiplying the three inequalities by 0, 5, and 1, respectively, and adding them up yields

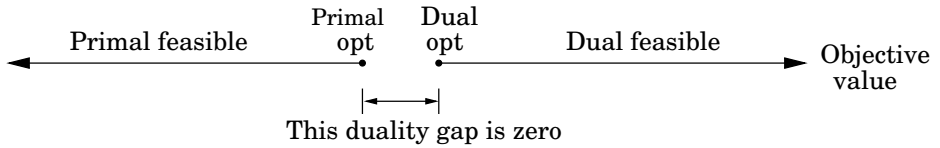
$$x_1 + 6x_2 \ \leq\ 1900.$$

So 1900 must indeed be the best possible value! The multipliers $(0, 5, 1)$ magically constitute a *certificate of optimality*! It is remarkable that such a certificate exists for this LP—and even if we knew there were one, how would we systematically go about finding it?

Let's investigate the issue by describing what we expect of these three multipliers, call them y_1, y_2, y_3.

Multiplier	Inequality
y_1	$x_1 \leq 200$
y_2	$x_2 \leq 300$
y_3	$x_1 + x_2 \leq 400$

Figure 7.9 By design, dual feasible values $\geq$ primal feasible values. The duality theorem tells us that moreover their optima coincide.



To start with, these y_i's must be nonnegative, for otherwise they are unqualified to multiply inequalities (multiplying an inequality by a negative number would flip the $\leq$ to $\geq$). After the multiplication and addition steps, we get the bound:

$$(y_1 + y_3)x_1 + (y_2 + y_3)x_2 \quad \leq \quad 200y_1 + 300y_2 + 400y_3.$$

We want the left-hand side to look like our objective function $x_1 + 6x_2$ so that the right-hand side is an upper bound on the optimum solution. For this we need $y_1 + y_3$ to be 1 and $y_2 + y_3$ to be 6. Come to think of it, it would be fine if $y_1 + y_3$ were larger than 1—the resulting certificate would be all the more convincing. Thus, we get an upper bound

$$x_1 + 6x_2 \quad \leq \quad 200y_1 + 300y_2 + 400y_3 \qquad \text{if} \qquad \left\{ \begin{array}{c} y_1, y_2, y_3 \geq 0 \\ y_1 + y_3 \geq 1 \\ y_2 + y_3 \geq 6 \end{array} \right\}.$$

We can easily find y's that satisfy the inequalities on the right by simply making them large enough, for example $(y_1, y_2, y_3) = (5, 3, 6)$. But these particular multipliers would tell us that the optimum solution of the LP is at most $200 \cdot 5 + 300 \cdot 3 + 400 \cdot 6 = 4300$, a bound that is far too loose to be of interest. What we want is a bound that is as tight as possible, so we should minimize $200y_1 + 300y_2 + 400y_3$ subject to the preceding inequalities. *And this is a new linear program*!

Therefore, finding the set of multipliers that gives the best upper bound on our original LP is tantamount to solving a new LP:

$$\begin{aligned} \min \ 200y_1 + 300y_2 + 400y_3 \\ y_1 + y_3 \geq 1 \\ y_2 + y_3 \geq 6 \\ y_1, y_2, y_3 \geq 0 \end{aligned}$$

By design, any feasible value of this *dual* LP is an upper bound on the original *primal* LP. So if we somehow find a pair of primal and dual feasible values that are equal, then they must both be optimal. Here is just such a pair:

$$\text{Primal} : (x_1, x_2) = (100, 300); \quad \text{Dual} : (y_1, y_2, y_3) = (0, 5, 1).$$

They both have value 1900, and therefore they certify each other's optimality (Figure 7.9).

Figure 7.10 A generic primal LP in matrix-vector form, and its dual.

Primal LP:

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

Dual LP:

$$\begin{aligned} \min\ & \mathbf{y}^T\mathbf{b} \\ & \mathbf{y}^T\mathbf{A} \geq \mathbf{c}^T \\ & \mathbf{y} \geq 0 \end{aligned}$$

Figure 7.11 In the most general case of linear programming, we have a set I of inequalities and a set E of equalities (a total of $m = |I| + |E|$ constraints) over n variables, of which a subset N are constrained to be nonnegative. The dual has $m = |I| + |E|$ variables, of which only those corresponding to I have nonnegativity constraints.

Primal LP:

$$\begin{aligned} \max\ & c_1x_1 + \cdots + c_nx_n \\ & a_{i1}x_1 + \cdots + a_{in}x_n \leq b_i \quad \text{for } i \in I \\ & a_{i1}x_1 + \cdots + a_{in}x_n = b_i \quad \text{for } i \in E \\ & x_j \geq 0 \quad \text{for } j \in N \end{aligned}$$

Dual LP:

$$\begin{aligned} \min\ & b_1y_1 + \cdots + b_my_m \\ & a_{1j}y_1 + \cdots + a_{mj}y_m \geq c_j \quad \text{for } j \in N \\ & a_{1j}y_1 + \cdots + a_{mj}y_m = c_j \quad \text{for } j \notin N \\ & y_i \geq 0 \quad \text{for } i \in I \end{aligned}$$

Amazingly, this is not just a lucky example, but a general phenomenon. To start with, the preceding construction—creating a multiplier for each primal constraint; writing a constraint in the dual for every variable of the primal, in which the sum is required to be above the objective coefficient of the corresponding primal variable; and optimizing the sum of the multipliers weighted by the primal right-hand sides—can be carried out for any LP, as shown in Figure 7.10, and in even greater generality in Figure 7.11. The second figure has one noteworthy addition: if the primal has an equality constraint, then the corresponding multiplier (or *dual variable*) need not be nonnegative, because the validity of equations is preserved when multiplied by negative numbers. So, the multipliers of equations are unrestricted variables. Notice also the simple symmetry between the two LPs, in that the matrix $A = (a_{ij})$ defines one primal constraint with each of its *rows*, and one dual constraint with each of its *columns*.

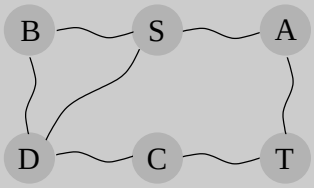
By construction, any feasible solution of the dual is an upper bound on any feasible solution of the primal. But moreover, their optima coincide!

Duality theorem *If a linear program has a bounded optimum, then so does its dual, and the two optimum values coincide.*

When the primal is the LP that expresses the max-flow problem, it is possible to assign interpretations to the dual variables that show the dual to be none other than the minimum-cut problem (Exercise 7.25). The relation between flows and cuts is therefore just a specific instance of the duality theorem. And in fact, the proof of this theorem falls out of the simplex algorithm, in much the same way as the max-flow min-cut theorem fell out of the analysis of the max-flow algorithm.

Visualizing duality

One can solve the shortest-path problem by the following "analog" device: Given a weighted undirected graph, build a *physical model* of it in which each edge is a string of length equal to the edge's weight, and each node is a knot at which the appropriate endpoints of strings are tied together. Then to find the shortest path from s to t, just *pull* s away from t until the gadget is taut. It is intuitively clear that this finds the shortest path from s to t.



There is nothing remarkable or surprising about all this until we notice the following: the shortest-path problem is a *minimization* problem, right? Then why are we *pulling* s away from t, an act whose purpose is, obviously, *maximization?* Answer: By pulling s away from t we solve *the dual* of the shortest-path problem! This dual has a very simple form (Exercise 7.28), with one variable x_u for each node u:

$$\begin{aligned} &\max\ x_S - x_T \\ &|x_u - x_v| \leq w_{uv} \quad \text{for all edges } \{u, v\} \end{aligned}$$

In words, the dual problem is to stretch s and t as far apart as possible, subject to the constraint that the endpoints of any edge $\{u, v\}$ are separated by a distance of at most w_{uv}.

7.5 Zero-sum games

We can represent various conflict situations in life by *matrix games*. For example, the schoolyard *rock-paper-scissors* game is specified by the *payoff matrix* illustrated here. There are two players, called Row and Column, and they each pick a move from $\{r, p, s\}$. They then look up the matrix entry corresponding to their moves, and Column pays Row this amount. It is Row's gain and Column's loss.

G =

		Column		
		r	p	s
Row	r	0	-1	1
	p	1	0	-1
	s	-1	1	0