Consider the system of equations formed by subtracting each equation in line (29.82) from the corresponding equation in line (29.80). The resulting system is

$$0 = (b_i - b'_i) - \sum_{j \in N} (a_{ij} - a'_{ij}) x_j \quad \text{for } i \in B$$

or, equivalently,

$$\sum_{j \in N} a_{ij} x_j = (b_i - b'_i) + \sum_{j \in N} a'_{ij} x_j \quad \text{for } i \in B \ .$$

Now, for each $i \in B$, apply Lemma 29.3 with $\alpha_j = a_{ij}$, $\beta_j = a'_{ij}$, $\gamma = b_i - b'_i$, and $I = N$. Since $\alpha_i = \beta_i$, we have that $a_{ij} = a'_{ij}$ for each $j \in N$, and since $\gamma = 0$, we have that $b_i = b'_i$. Thus, for the two slack forms, A and b are identical to A' and b'. Using a similar argument, Exercise 29.3-1 shows that it must also be the case that $c = c'$ and $v = v'$, and hence that the slack forms must be identical. ■

We can now show that cycling is the only possible reason that SIMPLEX might not terminate.

Lemma 29.5
If SIMPLEX fails to terminate in at most $\binom{n+m}{m}$ iterations, then it cycles.

Proof By Lemma 29.4, the set B of basic variables uniquely determines a slack form. There are $n + m$ variables and $|B| = m$, and therefore, there are at most $\binom{n+m}{m}$ ways to choose B. Thus, there are only at most $\binom{n+m}{m}$ unique slack forms. Therefore, if SIMPLEX runs for more than $\binom{n+m}{m}$ iterations, it must cycle. ■

Cycling is theoretically possible, but extremely rare. We can prevent it by choosing the entering and leaving variables somewhat more carefully. One option is to perturb the input slightly so that it is impossible to have two solutions with the same objective value. Another option is to break ties by always choosing the variable with the smallest index, a strategy known as ***Bland's rule***. We omit the proof that these strategies avoid cycling.

Lemma 29.6
If lines 4 and 9 of SIMPLEX always break ties by choosing the variable with the smallest index, then SIMPLEX must terminate. ■

We conclude this section with the following lemma.

Lemma 29.7
Assuming that INITIALIZE-SIMPLEX returns a slack form for which the basic solution is feasible, SIMPLEX either reports that a linear program is unbounded, or it terminates with a feasible solution in at most $\binom{n+m}{m}$ iterations.

Proof Lemmas 29.2 and 29.6 show that if INITIALIZE-SIMPLEX returns a slack form for which the basic solution is feasible, SIMPLEX either reports that a linear program is unbounded, or it terminates with a feasible solution. By the contrapositive of Lemma 29.5, if SIMPLEX terminates with a feasible solution, then it terminates in at most $\binom{n+m}{m}$ iterations. ■

Exercises

29.3-1
Complete the proof of Lemma 29.4 by showing that it must be the case that $c = c'$ and $\nu = \nu'$.

29.3-2
Show that the call to PIVOT in line 12 of SIMPLEX never decreases the value of ν.

29.3-3
Prove that the slack form given to the PIVOT procedure and the slack form that the procedure returns are equivalent.

29.3-4
Suppose we convert a linear program (A, b, c) in standard form to slack form. Show that the basic solution is feasible if and only if $b_i \geq 0$ for $i = 1, 2, \ldots, m$.

29.3-5
Solve the following linear program using SIMPLEX:

$$
\begin{aligned}
&\text{maximize} && 18x_1 + 12.5x_2 && \\
&\text{subject to} && && \\
& && x_1 + x_2 && \leq 20 \\
& && x_1 && \leq 12 \\
& && x_2 && \leq 16 \\
& && x_1, x_2 && \geq 0 \ .
\end{aligned}
$$

29.3-6

Solve the following linear program using SIMPLEX:

$$\begin{aligned}
&\text{maximize} && 5x_1 - 3x_2 \\
&\text{subject to} \\
&&& x_1 - x_2 \le 1 \\
&&& 2x_1 + x_2 \le 2 \\
&&& x_1, x_2 \ge 0 \ .
\end{aligned}$$

29.3-7

Solve the following linear program using SIMPLEX:

$$\begin{aligned}
&\text{minimize} && x_1 + x_2 + x_3 \\
&\text{subject to} \\
&&& 2x_1 + 7.5x_2 + 3x_3 \ge 10000 \\
&&& 20x_1 + 5x_2 + 10x_3 \ge 30000 \\
&&& x_1, x_2, x_3 \ge 0 \ .
\end{aligned}$$

29.3-8

In the proof of Lemma 29.5, we argued that there are at most $\binom{m+n}{n}$ ways to choose a set B of basic variables. Give an example of a linear program in which there are strictly fewer than $\binom{m+n}{n}$ ways to choose the set B.

29.4 Duality

We have proven that, under certain assumptions, SIMPLEX terminates. We have not yet shown that it actually finds an optimal solution to a linear program, however. In order to do so, we introduce a powerful concept called ***linear-programming duality***.

Duality enables us to prove that a solution is indeed optimal. We saw an example of duality in Chapter 26 with Theorem 26.6, the max-flow min-cut theorem. Suppose that, given an instance of a maximum-flow problem, we find a flow f with value $|f|$. How do we know whether f is a maximum flow? By the max-flow min-cut theorem, if we can find a cut whose value is also $|f|$, then we have verified that f is indeed a maximum flow. This relationship provides an example of duality: given a maximization problem, we define a related minimization problem such that the two problems have the same optimal objective values.

Given a linear program in which the objective is to maximize, we shall describe 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 a primal linear program in standard form, as in (29.16)–(29.18), we define the dual linear program as

$$\text{minimize} \quad \sum_{i=1}^{m} b_i y_i \tag{29.83}$$

subject to

$$\sum_{i=1}^{m} a_{ij} y_i \geq c_j \quad \text{for } j = 1, 2, \ldots, n , \tag{29.84}$$

$$y_i \geq 0 \quad \text{for } i = 1, 2, \ldots, m . \tag{29.85}$$

To form the dual, we change the maximization to a minimization, exchange the roles of coefficients on the right-hand sides and the objective function, and replace each less-than-or-equal-to by a greater-than-or-equal-to. Each of the m constraints in the primal has an associated variable y_i in the dual, and each of the n constraints in the dual has an associated variable x_j in the primal. For example, consider the linear program given in (29.53)–(29.57). The dual of this linear program is

$$\text{minimize} \quad 30y_1 + 24y_2 + 36y_3 \tag{29.86}$$

subject to

$$y_1 + 2y_2 + 4y_3 \geq 3 \tag{29.87}$$

$$y_1 + 2y_2 + y_3 \geq 1 \tag{29.88}$$

$$3y_1 + 5y_2 + 2y_3 \geq 2 \tag{29.89}$$

$$y_1, y_2, y_3 \geq 0 . \tag{29.90}$$

We shall show in Theorem 29.10 that the optimal value of the dual linear program is always equal to the optimal value of the primal linear program. Furthermore, the simplex algorithm actually implicitly solves both the primal and the dual linear programs simultaneously, thereby providing a proof of optimality.

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 29.8 (Weak linear-programming duality)
Let $\bar{x}$ be any feasible solution to the primal linear program in (29.16)–(29.18) and let $\bar{y}$ be any feasible solution to the dual linear program in (29.83)–(29.85). Then, we have

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

Proof We have

$$\begin{aligned}
\sum_{j=1}^{n} c_j \bar{x}_j &\le \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} \bar{y}_i \right) \bar{x}_j \quad \text{(by inequalities (29.84))} \\
&= \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} \bar{x}_j \right) \bar{y}_i \\
&\le \sum_{i=1}^{m} b_i \bar{y}_i \qquad \text{(by inequalities (29.17))} .
\end{aligned}$$

■

Corollary 29.9

Let $\bar{x}$ be a feasible solution to a primal linear program (A, b, c), and let $\bar{y}$ be a feasible solution to the corresponding dual linear program. 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 29.8, 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. ■

Before proving that there always is a dual solution whose value is equal to that of an optimal primal solution, we describe how to find such a solution. When we ran the simplex algorithm on the linear program in (29.53)–(29.57), the final iteration yielded the slack form (29.72)–(29.75) with objective $z = 28 - x_3/6 - x_5/6 - 2x_6/3$, $B = \{1, 2, 4\}$, and $N = \{3, 5, 6\}$. As we shall show below, the basic solution associated with the final slack form is indeed an optimal solution to the linear program; an optimal solution to linear program (29.53)–(29.57) is therefore $(\bar{x}_1, \bar{x}_2, \bar{x}_3) = (8, 4, 0)$, with objective value $(3 \cdot 8) + (1 \cdot 4) + (2 \cdot 0) = 28$. As we also show below, we can read off an optimal dual solution: the negatives of the coefficients of the primal objective function are the values of the dual variables. More precisely, suppose that the last slack form of the primal is

$$\begin{aligned}
z &= \nu' + \sum_{j \in N} c'_j x_j \\
x_i &= b'_i - \sum_{j \in N} a'_{ij} x_j \quad \text{for } i \in B .
\end{aligned}$$

Then, to produce an optimal dual solution, we set

$$\bar{y}_i = \begin{cases} -c'_{n+i} & \text{if } (n+i) \in N \ , \\ 0 & \text{otherwise} \ . \end{cases} \tag{29.91}$$

Thus, an optimal solution to the dual linear program defined in (29.86)–(29.90) is $\bar{y}_1 = 0$ (since $n + 1 = 4 \in B$), $\bar{y}_2 = -c'_5 = 1/6$, and $\bar{y}_3 = -c'_6 = 2/3$. Evaluating the dual objective function (29.86), we obtain an objective value of $(30 \cdot 0) + (24 \cdot (1/6)) + (36 \cdot (2/3)) = 28$, which confirms that the objective value of the primal is indeed equal to the objective value of the dual. Combining these calculations with Lemma 29.8 yields a proof that the optimal objective value of the primal linear program is 28. We now show that this approach applies in general: we can find an optimal solution to the dual and simultaneously prove that a solution to the primal is optimal.

Theorem 29.10 (Linear-programming duality)
Suppose that SIMPLEX returns values $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ for the primal linear program (A, b, c). Let N and B denote the nonbasic and basic variables for the final slack form, let c' denote the coefficients in the final slack form, and let $\bar{y} = (\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_m)$ be defined by equation (29.91). Then $\bar{x}$ is an optimal solution to the primal linear program, $\bar{y}$ is an optimal solution to the dual linear program, and

$$\sum_{j=1}^{n} c_j \bar{x}_j = \sum_{i=1}^{m} b_i \bar{y}_i \ . \tag{29.92}$$

Proof By Corollary 29.9, if we can find feasible solutions $\bar{x}$ and $\bar{y}$ that satisfy equation (29.92), then $\bar{x}$ and $\bar{y}$ must be optimal primal and dual solutions. We shall now show that the solutions $\bar{x}$ and $\bar{y}$ described in the statement of the theorem satisfy equation (29.92).

Suppose that we run SIMPLEX on a primal linear program, as given in lines (29.16)–(29.18). The algorithm proceeds through a series of slack forms until it terminates with a final slack form with objective function

$$z = v' + \sum_{j \in N} c'_j x_j \ . \tag{29.93}$$

Since SIMPLEX terminated with a solution, by the condition in line 3 we know that

$$c'_j \le 0 \quad \text{for all } j \in N \ . \tag{29.94}$$

If we define

$$c'_j = 0 \quad \text{for all } j \in B\ , \tag{29.95}$$

we can rewrite equation (29.93) as

$$\begin{aligned} z &= v' + \sum_{j \in N} c'_j x_j \\ &= v' + \sum_{j \in N} c'_j x_j + \sum_{j \in B} c'_j x_j \quad (\text{because } c'_j = 0 \text{ if } j \in B) \\ &= v' + \sum_{j=1}^{n+m} c'_j x_j \qquad\qquad (\text{because } N \cup B = \{1, 2, \ldots, n+m\})\ . \end{aligned} \tag{29.96}$$

For the basic solution $\bar{x}$ associated with this final slack form, $\bar{x}_j = 0$ for all $j \in N$, and $z = v'$. Since all slack forms are equivalent, if we evaluate the original objective function on $\bar{x}$, we must obtain the same objective value:

$$\begin{aligned} \sum_{j=1}^{n} c_j \bar{x}_j &= v' + \sum_{j=1}^{n+m} c'_j \bar{x}_j && (29.97) \\ &= v' + \sum_{j \in N} c'_j \bar{x}_j + \sum_{j \in B} c'_j \bar{x}_j \\ &= v' + \sum_{j \in N} (c'_j \cdot 0) + \sum_{j \in B} (0 \cdot \bar{x}_j) && (29.98) \\ &= v'\ . \end{aligned}$$

We shall now show that $\bar{y}$, defined by equation (29.91), is feasible for the dual linear program and that its objective value $\sum_{i=1}^{m} b_i \bar{y}_i$ equals $\sum_{j=1}^{n} c_j \bar{x}_j$. Equation (29.97) says that the first and last slack forms, evaluated at $\bar{x}$, are equal. More generally, the equivalence of all slack forms implies that for *any* set of values $x = (x_1, x_2, \ldots, x_n)$, we have

$$\sum_{j=1}^{n} c_j x_j = v' + \sum_{j=1}^{n+m} c'_j x_j\ .$$

Therefore, for any particular set of values $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$, we have

$$
\begin{aligned}
&\sum_{j=1}^{n} c_j \bar{x}_j \\
&= \nu' + \sum_{j=1}^{n+m} c'_j \bar{x}_j \\
&= \nu' + \sum_{j=1}^{n} c'_j \bar{x}_j + \sum_{j=n+1}^{n+m} c'_j \bar{x}_j \\
&= \nu' + \sum_{j=1}^{n} c'_j \bar{x}_j + \sum_{i=1}^{m} c'_{n+i} \bar{x}_{n+i} \\
&= \nu' + \sum_{j=1}^{n} c'_j \bar{x}_j + \sum_{i=1}^{m} (-\bar{y}_i)\, \bar{x}_{n+i} && \text{(by equations (29.91) and (29.95))} \\
&= \nu' + \sum_{j=1}^{n} c'_j \bar{x}_j + \sum_{i=1}^{m} (-\bar{y}_i) \left(b_i - \sum_{j=1}^{n} a_{ij} \bar{x}_j \right) && \text{(by equation (29.32))} \\
&= \nu' + \sum_{j=1}^{n} c'_j \bar{x}_j - \sum_{i=1}^{m} b_i \bar{y}_i + \sum_{i=1}^{m} \sum_{j=1}^{n} (a_{ij} \bar{x}_j)\, \bar{y}_i \\
&= \nu' + \sum_{j=1}^{n} c'_j \bar{x}_j - \sum_{i=1}^{m} b_i \bar{y}_i + \sum_{j=1}^{n} \sum_{i=1}^{m} (a_{ij} \bar{y}_i)\, \bar{x}_j \\
&= \left(\nu' - \sum_{i=1}^{m} b_i \bar{y}_i \right) + \sum_{j=1}^{n} \left(c'_j + \sum_{i=1}^{m} a_{ij} \bar{y}_i \right) \bar{x}_j \,,
\end{aligned}
$$

so that

$$
\sum_{j=1}^{n} c_j \bar{x}_j = \left(\nu' - \sum_{i=1}^{m} b_i \bar{y}_i \right) + \sum_{j=1}^{n} \left(c'_j + \sum_{i=1}^{m} a_{ij} \bar{y}_i \right) \bar{x}_j \,. \tag{29.99}
$$

Applying Lemma 29.3 to equation (29.99), we obtain

$$
\nu' - \sum_{i=1}^{m} b_i \bar{y}_i = 0 \,, \tag{29.100}
$$

$$
c'_j + \sum_{i=1}^{m} a_{ij} \bar{y}_i = c_j \quad \text{for } j = 1, 2, \ldots, n \,. \tag{29.101}
$$

By equation (29.100), we have that $\sum_{i=1}^{m} b_i \bar{y}_i = \nu'$, and hence the objective value of the dual $\left(\sum_{i=1}^{m} b_i \bar{y}_i \right)$ is equal to that of the primal (ν'). It remains to show

that the solution $\bar{y}$ is feasible for the dual problem. From inequalities (29.94) and equations (29.95), we have that $c'_j \le 0$ for all $j = 1, 2, \ldots, n+m$. Hence, for any $j = 1, 2, \ldots, n$, equations (29.101) imply that

$$\begin{aligned} c_j &= c'_j + \sum_{i=1}^{m} a_{ij}\bar{y}_i \\ &\le \sum_{i=1}^{m} a_{ij}\bar{y}_i \ , \end{aligned}$$

which satisfies the constraints (29.84) of the dual. Finally, since $c'_j \le 0$ for each $j \in N \cup B$, when we set $\bar{y}$ according to equation (29.91), we have that each $\bar{y}_i \ge 0$, and so the nonnegativity constraints are satisfied as well. ■

We have shown that, given a feasible linear program, if INITIALIZE-SIMPLEX returns a feasible solution, and if SIMPLEX terminates without returning "unbounded," then the solution returned is indeed an optimal solution. We have also shown how to construct an optimal solution to the dual linear program.

Exercises

29.4-1
Formulate the dual of the linear program given in Exercise 29.3-5.

29.4-2
Suppose that we have a linear program that is not in standard form. We could produce the dual by first converting it to standard form, and then taking the dual. It would be more convenient, however, to be able to produce the dual directly. Explain how we can directly take the dual of an arbitrary linear program.

29.4-3
Write down the dual of the maximum-flow linear program, as given in lines (29.47)–(29.50) on page 860. Explain how to interpret this formulation as a minimum-cut problem.

29.4-4
Write down the dual of the minimum-cost-flow linear program, as given in lines (29.51)–(29.52) on page 862. Explain how to interpret this problem in terms of graphs and flows.

29.4-5
Show that the dual of the dual of a linear program is the primal linear program.

29.4-6
Which result from Chapter 26 can be interpreted as weak duality for the maximum-flow problem?

29.5 The initial basic feasible solution

In this section, we first describe how to test whether a linear program is feasible, and if it is, how to produce a slack form for which the basic solution is feasible. We conclude by proving the fundamental theorem of linear programming, which says that the SIMPLEX procedure always produces the correct result.

Finding an initial solution

In Section 29.3, we assumed that we had a procedure INITIALIZE-SIMPLEX that determines whether a linear program has any feasible solutions, and if it does, gives a slack form for which the basic solution is feasible. We describe this procedure here.

A linear program can be feasible, yet the initial basic solution might not be feasible. Consider, for example, the following linear program:

$$\begin{aligned}
&\text{maximize} && 2x_1 - x_2 && && && (29.102)\\
&\text{subject to} && && && && \\
& && 2x_1 - x_2 && \le && 2 && (29.103)\\
& && x_1 - 5x_2 && \le && -4 && (29.104)\\
& && x_1, x_2 && \ge && 0\ . && (29.105)
\end{aligned}$$

If we were to convert this linear program to slack form, the basic solution would set $x_1 = 0$ and $x_2 = 0$. This solution violates constraint (29.104), and so it is not a feasible solution. Thus, INITIALIZE-SIMPLEX cannot just return the obvious slack form. In order to determine whether a linear program has any feasible solutions, we will formulate an ***auxiliary linear program***. For this auxiliary linear program, we can find (with a little work) a slack form for which the basic solution is feasible. Furthermore, the solution of this auxiliary linear program determines whether the initial linear program is feasible and if so, it provides a feasible solution with which we can initialize SIMPLEX.

Lemma 29.11
Let L be a linear program in standard form, given as in (29.16)–(29.18). Let x_0 be a new variable, and let L_{aux} be the following linear program with $n + 1$ variables:

$$\text{maximize} \qquad -x_0 \tag{29.106}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j - x_0 \leq b_i \quad \text{for } i = 1, 2, \ldots, m\,, \tag{29.107}$$

$$x_j \geq 0 \quad \text{for } j = 0, 1, \ldots, n\,. \tag{29.108}$$

Then L is feasible if and only if the optimal objective value of L_{aux} is 0.

Proof Suppose that L has a feasible solution $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$. Then the solution $\bar{x}_0 = 0$ combined with $\bar{x}$ is a feasible solution to L_{aux} with objective value 0. Since $x_0 \geq 0$ is a constraint of L_{aux} and the objective function is to maximize $-x_0$, this solution must be optimal for L_{aux}.

Conversely, suppose that the optimal objective value of L_{aux} is 0. Then $\bar{x}_0 = 0$, and the remaining solution values of $\bar{x}$ satisfy the constraints of L. ■

We now describe our strategy to find an initial basic feasible solution for a linear program L in standard form:

INITIALIZE-SIMPLEX(A, b, c)

```
1  let k be the index of the minimum b_i
2  if b_k ≥ 0                    // is the initial basic solution feasible?
3      return ({1, 2, ..., n}, {n + 1, n + 2, ..., n + m}, A, b, c, 0)
4  form L_aux by adding −x_0 to the left-hand side of each constraint
       and setting the objective function to −x_0
5  let (N, B, A, b, c, v) be the resulting slack form for L_aux
6  l = n + k
7  // L_aux has n + 1 nonbasic variables and m basic variables.
8  (N, B, A, b, c, v) = PIVOT(N, B, A, b, c, v, l, 0)
9  // The basic solution is now feasible for L_aux.
10 iterate the while loop of lines 3–12 of SIMPLEX until an optimal solution
       to L_aux is found
11 if the optimal solution to L_aux sets x̄_0 to 0
12     if x̄_0 is basic
13         perform one (degenerate) pivot to make it nonbasic
14     from the final slack form of L_aux, remove x_0 from the constraints and
           restore the original objective function of L, but replace each basic
           variable in this objective function by the right-hand side of its
           associated constraint
15     return the modified final slack form
16 else return "infeasible"
```

INITIALIZE-SIMPLEX works as follows. In lines 1–3, we implicitly test the basic solution to the initial slack form for L given by $N = \{1, 2, \ldots, n\}$, $B = \{n+1, n+2, \ldots, n+m\}$, $\bar{x}_i = b_i$ for all $i \in B$, and $\bar{x}_j = 0$ for all $j \in N$. (Creating the slack form requires no explicit effort, as the values of A, b, and c are the same in both slack and standard forms.) If line 2 finds this basic solution to be feasible—that is, $\bar{x}_i \geq 0$ for all $i \in N \cup B$—then line 3 returns the slack form. Otherwise, in line 4, we form the auxiliary linear program L_{aux} as in Lemma 29.11. Since the initial basic solution to L is not feasible, the initial basic solution to the slack form for L_{aux} cannot be feasible either. To find a basic feasible solution, we perform a single pivot operation. Line 6 selects $l = n + k$ as the index of the basic variable that will be the leaving variable in the upcoming pivot operation. Since the basic variables are $x_{n+1}, x_{n+2}, \ldots, x_{n+m}$, the leaving variable x_l will be the one with the most negative value. Line 8 performs that call of PIVOT, with x_0 entering and x_l leaving. We shall see shortly that the basic solution resulting from this call of PIVOT will be feasible. Now that we have a slack form for which the basic solution is feasible, we can, in line 10, repeatedly call PIVOT to fully solve the auxiliary linear program. As the test in line 11 demonstrates, if we find an optimal solution to L_{aux} with objective value 0, then in lines 12–14, we create a slack form for L for which the basic solution is feasible. To do so, we first, in lines 12–13, handle the degenerate case in which x_0 may still be basic with value $\bar{x}_0 = 0$. In this case, we perform a pivot step to remove x_0 from the basis, using any $e \in N$ such that $a_{0e} \neq 0$ as the entering variable. The new basic solution remains feasible; the degenerate pivot does not change the value of any variable. Next we delete all x_0 terms from the constraints and restore the original objective function for L. The original objective function may contain both basic and nonbasic variables. Therefore, in the objective function we replace each basic variable by the right-hand side of its associated constraint. Line 15 then returns this modified slack form. If, on the other hand, line 11 discovers that the original linear program L is infeasible, then line 16 returns this information.

We now demonstrate the operation of INITIALIZE-SIMPLEX on the linear program (29.102)–(29.105). This linear program is feasible if we can find nonnegative values for x_1 and x_2 that satisfy inequalities (29.103) and (29.104). Using Lemma 29.11, we formulate the auxiliary linear program

$$\begin{aligned}
&\text{maximize} && -x_0 && && && && && && (29.109)\\
&\text{subject to} \\
&& 2x_1 &- x_2 &- x_0 &\leq 2 && && && && (29.110)\\
&& x_1 &- 5x_2 &- x_0 &\leq -4 && && && && (29.111)\\
&& x_1, x_2, x_0 & && &\geq 0\ . && && && &
\end{aligned}$$

By Lemma 29.11, if the optimal objective value of this auxiliary linear program is 0, then the original linear program has a feasible solution. If the optimal objective

value of this auxiliary linear program is negative, then the original linear program does not have a feasible solution.

We write this linear program in slack form, obtaining

$$\begin{aligned} z &= && && && - && x_0 \\ x_3 &= && 2 && - && 2x_1 && + && x_2 && + && x_0 \\ x_4 &= && -4 && - && x_1 && + && 5x_2 && + && x_0 \ . \end{aligned}$$

We are not out of the woods yet, because the basic solution, which would set $x_4 = -4$, is not feasible for this auxiliary linear program. We can, however, with one call to PIVOT, convert this slack form into one in which the basic solution is feasible. As line 8 indicates, we choose x_0 to be the entering variable. In line 6, we choose as the leaving variable x_4, which is the basic variable whose value in the basic solution is most negative. After pivoting, we have the slack form

$$\begin{aligned} z &= && -4 && - && x_1 && + && 5x_2 && - && x_4 \\ x_0 &= && 4 && + && x_1 && - && 5x_2 && + && x_4 \\ x_3 &= && 6 && - && x_1 && - && 4x_2 && + && x_4 \ . \end{aligned}$$

The associated basic solution is $(\bar{x}_0, \bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4) = (4, 0, 0, 6, 0)$, which is feasible. We now repeatedly call PIVOT until we obtain an optimal solution to L_{aux}. In this case, one call to PIVOT with x_2 entering and x_0 leaving yields

$$\begin{aligned} z &= && && - && x_0 \\ x_2 &= && \frac{4}{5} && - && \frac{x_0}{5} && + && \frac{x_1}{5} && + && \frac{x_4}{5} \\ x_3 &= && \frac{14}{5} && + && \frac{4x_0}{5} && - && \frac{9x_1}{5} && + && \frac{x_4}{5} \ . \end{aligned}$$

This slack form is the final solution to the auxiliary problem. Since this solution has $x_0 = 0$, we know that our initial problem was feasible. Furthermore, since $x_0 = 0$, we can just remove it from the set of constraints. We then restore the original objective function, with appropriate substitutions made to include only nonbasic variables. In our example, we get the objective function

$$2x_1 - x_2 = 2x_1 - \left(\frac{4}{5} - \frac{x_0}{5} + \frac{x_1}{5} + \frac{x_4}{5}\right) \ .$$

Setting $x_0 = 0$ and simplifying, we get the objective function

$$-\frac{4}{5} + \frac{9x_1}{5} - \frac{x_4}{5} \ ,$$

and the slack form

$$\begin{aligned} z &= -\frac{4}{5} + \frac{9x_1}{5} - \frac{x_4}{5} \\ x_2 &= \frac{4}{5} + \frac{x_1}{5} + \frac{x_4}{5} \\ x_3 &= \frac{14}{5} - \frac{9x_1}{5} + \frac{x_4}{5} \; . \end{aligned}$$

This slack form has a feasible basic solution, and we can return it to procedure SIMPLEX.

We now formally show the correctness of INITIALIZE-SIMPLEX.

Lemma 29.12

If a linear program L has no feasible solution, then INITIALIZE-SIMPLEX returns "infeasible." Otherwise, it returns a valid slack form for which the basic solution is feasible.

Proof First suppose that the linear program L has no feasible solution. Then by Lemma 29.11, the optimal objective value of L_{aux}, defined in (29.106)–(29.108), is nonzero, and by the nonnegativity constraint on x_0, the optimal objective value must be negative. Furthermore, this objective value must be finite, since setting $x_i = 0$, for $i = 1, 2, \ldots, n$, and $x_0 = |\min_{i=1}^{m} \{b_i\}|$ is feasible, and this solution has objective value $-|\min_{i=1}^{m} \{b_i\}|$. Therefore, line 10 of INITIALIZE-SIMPLEX finds a solution with a nonpositive objective value. Let $\bar{x}$ be the basic solution associated with the final slack form. We cannot have $\bar{x}_0 = 0$, because then L_{aux} would have objective value 0, which contradicts that the objective value is negative. Thus the test in line 11 results in line 16 returning "infeasible."

Suppose now that the linear program L does have a feasible solution. From Exercise 29.3-4, we know that if $b_i \geq 0$ for $i = 1, 2, \ldots, m$, then the basic solution associated with the initial slack form is feasible. In this case, lines 2–3 return the slack form associated with the input. (Converting the standard form to slack form is easy, since A, b, and c are the same in both.)

In the remainder of the proof, we handle the case in which the linear program is feasible but we do not return in line 3. We argue that in this case, lines 4–10 find a feasible solution to L_{aux} with objective value 0. First, by lines 1–2, we must have

$$b_k < 0 \; ,$$

and

$$b_k \leq b_i \quad \text{for each } i \in B \; . \tag{29.112}$$

In line 8, we perform one pivot operation in which the leaving variable x_l (recall that $l = n + k$, so that $b_l < 0$) is the left-hand side of the equation with minimum b_i, and the entering variable is x_0, the extra added variable. We now show

that after this pivot, all entries of b are nonnegative, and hence the basic solution to L_{aux} is feasible. Letting $\bar{x}$ be the basic solution after the call to PIVOT, and letting $\widehat{b}$ and $\widehat{B}$ be values returned by PIVOT, Lemma 29.1 implies that

$$\bar{x}_i = \begin{cases} b_i - a_{ie}\widehat{b}_e & \text{if } i \in \widehat{B} - \{e\} \,, \\ b_l / a_{le} & \text{if } i = e \,. \end{cases} \tag{29.113}$$

The call to PIVOT in line 8 has $e = 0$. If we rewrite inequalities (29.107), to include coefficients a_{i0},

$$\sum_{j=0}^{n} a_{ij} x_j \le b_i \quad \text{for } i = 1, 2, \ldots, m \,, \tag{29.114}$$

then

$$a_{i0} = a_{ie} = -1 \quad \text{for each } i \in B \,. \tag{29.115}$$

(Note that a_{i0} is the coefficient of x_0 as it appears in inequalities (29.114), not the negation of the coefficient, because L_{aux} is in standard rather than slack form.) Since $l \in B$, we also have that $a_{le} = -1$. Thus, $b_l / a_{le} > 0$, and so $\bar{x}_e > 0$. For the remaining basic variables, we have

$$\begin{aligned} \bar{x}_i &= b_i - a_{ie}\widehat{b}_e && \text{(by equation (29.113))} \\ &= b_i - a_{ie}(b_l / a_{le}) && \text{(by line 3 of PIVOT)} \\ &= b_i - b_l && \text{(by equation (29.115) and } a_{le} = -1) \\ &\ge 0 && \text{(by inequality (29.112)) ,} \end{aligned}$$

which implies that each basic variable is now nonnegative. Hence the basic solution after the call to PIVOT in line 8 is feasible. We next execute line 10, which solves L_{aux}. Since we have assumed that L has a feasible solution, Lemma 29.11 implies that L_{aux} has an optimal solution with objective value 0. Since all the slack forms are equivalent, the final basic solution to L_{aux} must have $\bar{x}_0 = 0$, and after removing x_0 from the linear program, we obtain a slack form that is feasible for L. Line 15 then returns this slack form. ■

Fundamental theorem of linear programming

We conclude this chapter by showing that the SIMPLEX procedure works. In particular, any linear program either is infeasible, is unbounded, or has an optimal solution with a finite objective value. In each case, SIMPLEX acts appropriately.

Theorem 29.13 (Fundamental theorem of linear programming)
Any linear program L, given in standard form, either

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

If L is infeasible, SIMPLEX returns "infeasible." If L is unbounded, SIMPLEX returns "unbounded." Otherwise, SIMPLEX returns an optimal solution with a finite objective value.

Proof By Lemma 29.12, if linear program L is infeasible, then SIMPLEX returns "infeasible." Now suppose that the linear program L is feasible. By Lemma 29.12, INITIALIZE-SIMPLEX returns a slack form for which the basic solution is feasible. By Lemma 29.7, therefore, SIMPLEX either returns "unbounded" or terminates with a feasible solution. If it terminates with a finite solution, then Theorem 29.10 tells us that this solution is optimal. On the other hand, if SIMPLEX returns "unbounded," Lemma 29.2 tells us the linear program L is indeed unbounded. Since SIMPLEX always terminates in one of these ways, the proof is complete. ■

Exercises

29.5-1
Give detailed pseudocode to implement lines 5 and 14 of INITIALIZE-SIMPLEX.

29.5-2
Show that when the main loop of SIMPLEX is run by INITIALIZE-SIMPLEX, it can never return "unbounded."

29.5-3
Suppose that we are given a linear program L in standard form, and suppose that for both L and the dual of L, the basic solutions associated with the initial slack forms are feasible. Show that the optimal objective value of L is 0.

29.5-4
Suppose that we allow strict inequalities in a linear program. Show that in this case, the fundamental theorem of linear programming does not hold.

29.5-5

Solve the following linear program using SIMPLEX:

$$\begin{aligned}
\text{maximize} \quad & x_1 + 3x_2 \\
\text{subject to} \quad & \\
& x_1 - x_2 \le 8 \\
& -x_1 - x_2 \le -3 \\
& -x_1 + 4x_2 \le 2 \\
& x_1, x_2 \ge 0 \ .
\end{aligned}$$

29.5-6

Solve the following linear program using SIMPLEX:

$$\begin{aligned}
\text{maximize} \quad & x_1 - 2x_2 \\
\text{subject to} \quad & \\
& x_1 + 2x_2 \le 4 \\
& -2x_1 - 6x_2 \le -12 \\
& x_2 \le 1 \\
& x_1, x_2 \ge 0 \ .
\end{aligned}$$

29.5-7

Solve the following linear program using SIMPLEX:

$$\begin{aligned}
\text{maximize} \quad & x_1 + 3x_2 \\
\text{subject to} \quad & \\
& -x_1 + x_2 \le -1 \\
& -x_1 - x_2 \le -3 \\
& -x_1 + 4x_2 \le 2 \\
& x_1, x_2 \ge 0 \ .
\end{aligned}$$

29.5-8

Solve the linear program given in (29.6)–(29.10).

29.5-9

Consider the following 1-variable linear program, which we call P:

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

where r, s, and t are arbitrary real numbers. Let D be the dual of P.

State for which values of r, s, and t you can assert that

1. Both P and D have optimal solutions with finite objective values.
2. P is feasible, but D is infeasible.
3. D is feasible, but P is infeasible.
4. Neither P nor D is feasible.

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. Show that if we have an algorithm for linear programming, we can 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. Show that if we have an algorithm for the linear-inequality feasibility problem, we can 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 (29.16)–(29.18), and let $\bar{y}$ be a feasible solution to the dual linear program given in (29.83)–(29.85). Complementary slackness states that the following conditions are necessary and sufficient for $\bar{x}$ and $\bar{y}$ to be optimal:

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

and

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

a. Verify that complementary slackness holds for the linear program in lines (29.53)–(29.57).

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 (29.16)–(29.18) 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 (29.83)–(29.85),
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 integral values. Exercise 34.5-3 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 29.8) holds for an integer linear program.

b. Show that duality (Theorem 29.10) does not always hold for an integer linear program.

c. Given a primal linear program in standard form, let us define P to be the optimal objective value for the primal linear program, D to be the optimal objective value for its dual, IP to 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 to 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

Let A be an $m \times n$ matrix and c be an n-vector. Then Farkas's lemma states that exactly one of the systems

$$\begin{aligned} Ax &\leq 0\,, \\ c^{\mathrm{T}}x &> 0 \end{aligned}$$

and

$$\begin{aligned} A^{\mathrm{T}}y &= c\,, \\ y &\geq 0 \end{aligned}$$

is solvable, where x is an n-vector and y is an m-vector. Prove Farkas's lemma.

29-5 Minimum-cost circulation

In this problem, we consider a variant of the minimum-cost-flow problem from Section 29.2 in which we are not given a demand, a source, or a sink. Instead, we are given, as before, a flow network 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 $a(u, v) > 0$. Characterize an optimal solution to the minimum-cost-circulation problem.

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 discern 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 [69], Gass [130], Karloff [197], Schrijver [303], and Vanderbei [344]. Many other books give a good coverage of linear programming, including those by Papadimitriou and Steiglitz [271] 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 [69] and [197].

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 [154] 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 [198] 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 [141] and the book by Ye [361].

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 very 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 [322] 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 [268] and the citations therein.

30 Polynomials and the FFT

The straightforward method of adding two polynomials of degree n takes $\Theta(n)$ time, but the straightforward method of multiplying them takes $\Theta(n^2)$ time. In this chapter, we shall show how the fast Fourier transform, or FFT, can reduce the time to multiply polynomials to $\Theta(n \lg n)$.

The most common use for Fourier transforms, and hence the FFT, is in signal processing. A signal is given in the ***time domain***: as a function mapping time to amplitude. Fourier analysis allows us to express the signal as a weighted sum of phase-shifted sinusoids of varying frequencies. The weights and phases associated with the frequencies characterize the signal in the ***frequency domain***. Among the many everyday applications of FFT's are compression techniques used to encode digital video and audio information, including MP3 files. Several fine books delve into the rich area of signal processing; the chapter notes reference a few of them.

Polynomials

A ***polynomial*** in the variable x over an algebraic field F represents a function $A(x)$ as a formal sum:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \ .$$

We call the values $a_0, a_1, \ldots, a_{n-1}$ the ***coefficients*** of the polynomial. The coefficients are drawn from a field F, typically the set $\mathbb{C}$ of complex numbers. A polynomial $A(x)$ has ***degree*** k if its highest nonzero coefficient is a_k; we write that degree$(A) = k$. Any integer strictly greater than the degree of a polynomial is a ***degree-bound*** of that polynomial. Therefore, the degree of a polynomial of degree-bound n may be any integer between 0 and $n - 1$, inclusive.

We can define a variety of operations on polynomials. For ***polynomial addition***, if $A(x)$ and $B(x)$ are polynomials of degree-bound n, their ***sum*** is a polyno-

mial $C(x)$, also of degree-bound n, such that $C(x) = A(x) + B(x)$ for all x in the underlying field. That is, if

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

and

$$B(x) = \sum_{j=0}^{n-1} b_j x^j ,$$

then

$$C(x) = \sum_{j=0}^{n-1} c_j x^j ,$$

where $c_j = a_j + b_j$ for $j = 0, 1, \ldots, n-1$. For example, if we have the polynomials $A(x) = 6x^3 + 7x^2 - 10x + 9$ and $B(x) = -2x^3 + 4x - 5$, then $C(x) = 4x^3 + 7x^2 - 6x + 4$.

For ***polynomial multiplication***, if $A(x)$ and $B(x)$ are polynomials of degree-bound n, their ***product*** $C(x)$ is a polynomial of degree-bound $2n-1$ such that $C(x) = A(x)B(x)$ for all x in the underlying field. You probably have multiplied polynomials before, by multiplying each term in $A(x)$ by each term in $B(x)$ and then combining terms with equal powers. For example, we can multiply $A(x) = 6x^3 + 7x^2 - 10x + 9$ and $B(x) = -2x^3 + 4x - 5$ as follows:

$$\begin{array}{rrrrrrrrr}
 & & & & 6x^3 & + & 7x^2 & - 10x & + & 9 \\
 & & & - & 2x^3 & & & + 4x & - & 5 \\
\hline
 & & & - & 30x^3 & - & 35x^2 & + 50x & - & 45 \\
 & & 24x^4 & + & 28x^3 & - & 40x^2 & + 36x & & \\
- 12x^6 & - & 14x^5 & + & 20x^4 & - & 18x^3 & & & \\
\hline
- 12x^6 & - & 14x^5 & + & 44x^4 & - & 20x^3 & - 75x^2 & + 86x & - 45
\end{array}$$

Another way to express the product $C(x)$ is

$$C(x) = \sum_{j=0}^{2n-2} c_j x^j , \tag{30.1}$$

where

$$c_j = \sum_{k=0}^{j} a_k b_{j-k} . \tag{30.2}$$

Note that degree(C) = degree(A) + degree(B), implying that if A is a polynomial of degree-bound n_a and B is a polynomial of degree-bound n_b, then C is a polynomial of degree-bound $n_a + n_b - 1$. Since a polynomial of degree-bound k is also a polynomial of degree-bound $k + 1$, we will normally say that the product polynomial C is a polynomial of degree-bound $n_a + n_b$.

Chapter outline

Section 30.1 presents two ways to represent polynomials: the coefficient representation and the point-value representation. The straightforward methods for multiplying polynomials—equations (30.1) and (30.2)—take $\Theta(n^2)$ time when we represent polynomials in coefficient form, but only $\Theta(n)$ time when we represent them in point-value form. We can, however, multiply polynomials using the coefficient representation in only $\Theta(n \lg n)$ time by converting between the two representations. To see why this approach works, we must first study complex roots of unity, which we do in Section 30.2. Then, we use the FFT and its inverse, also described in Section 30.2, to perform the conversions. Section 30.3 shows how to implement the FFT quickly in both serial and parallel models.

This chapter uses complex numbers extensively, and within this chapter we use the symbol i exclusively to denote $\sqrt{-1}$.

30.1 Representing polynomials

The coefficient and point-value representations of polynomials are in a sense equivalent; that is, a polynomial in point-value form has a unique counterpart in coefficient form. In this section, we introduce the two representations and show how to combine them so that we can multiply two degree-bound n polynomials in $\Theta(n \lg n)$ time.

Coefficient representation

A ***coefficient representation*** of a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$ of degree-bound n is a vector of coefficients $a = (a_0, a_1, \ldots, a_{n-1})$. In matrix equations in this chapter, we shall generally treat vectors as column vectors.

The coefficient representation is convenient for certain operations on polynomials. For example, the operation of ***evaluating*** the polynomial $A(x)$ at a given point x_0 consists of computing the value of $A(x_0)$. We can evaluate a polynomial in $\Theta(n)$ time using ***Horner's rule***:

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-2} + x_0(a_{n-1}))\cdots)) \, .$$

Similarly, adding two polynomials represented by the coefficient vectors $a = (a_0, a_1, \ldots, a_{n-1})$ and $b = (b_0, b_1, \ldots, b_{n-1})$ takes $\Theta(n)$ time: we just produce the coefficient vector $c = (c_0, c_1, \ldots, c_{n-1})$, where $c_j = a_j + b_j$ for $j = 0, 1, \ldots, n-1$.

Now, consider multiplying two degree-bound n polynomials $A(x)$ and $B(x)$ represented in coefficient form. If we use the method described by equations (30.1) and (30.2), multiplying polynomials takes time $\Theta(n^2)$, since we must multiply each coefficient in the vector a by each coefficient in the vector b. The operation of multiplying polynomials in coefficient form seems to be considerably more difficult than that of evaluating a polynomial or adding two polynomials. The resulting coefficient vector c, given by equation (30.2), is also called the ***convolution*** of the input vectors a and b, denoted $c = a \otimes b$. Since multiplying polynomials and computing convolutions are fundamental computational problems of considerable practical importance, this chapter concentrates on efficient algorithms for them.

Point-value representation

A ***point-value representation*** of a polynomial $A(x)$ of degree-bound n is a set of n ***point-value pairs***

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

such that all of the x_k are distinct and

$$y_k = A(x_k) \tag{30.3}$$

for $k = 0, 1, \ldots, n-1$. A polynomial has many different point-value representations, since we can use any set of n distinct points $x_0, x_1, \ldots, x_{n-1}$ as a basis for the representation.

Computing a point-value representation for a polynomial given in coefficient form is in principle straightforward, since all we have to do is select n distinct points $x_0, x_1, \ldots, x_{n-1}$ and then evaluate $A(x_k)$ for $k = 0, 1, \ldots, n-1$. With Horner's method, evaluating a polynomial at n points takes time $\Theta(n^2)$. We shall see later that if we choose the points x_k cleverly, we can accelerate this computation to run in time $\Theta(n \lg n)$.

The inverse of evaluation—determining the coefficient form of a polynomial from a point-value representation—is ***interpolation***. The following theorem shows that interpolation is well defined when the desired interpolating polynomial must have a degree-bound equal to the given number of point-value pairs.

Theorem 30.1 (Uniqueness of an interpolating polynomial)
For any set $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ of n point-value pairs such that all the x_k values are distinct, there is a unique polynomial $A(x)$ of degree-bound n such that $y_k = A(x_k)$ for $k = 0, 1, \ldots, n-1$.

Proof The proof relies on the existence of the inverse of a certain matrix. Equation (30.3) is equivalent to the matrix equation

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} . \tag{30.4}$$

The matrix on the left is denoted $V(x_0, x_1, \ldots, x_{n-1})$ and is known as a Vandermonde matrix. By Problem D-1, this matrix has determinant

$$\prod_{0 \le j < k \le n-1} (x_k - x_j) ,$$

and therefore, by Theorem D.5, it is invertible (that is, nonsingular) if the x_k are distinct. Thus, we can solve for the coefficients a_j uniquely given the point-value representation:

$$a = V(x_0, x_1, \ldots, x_{n-1})^{-1} y . \qquad \blacksquare$$

The proof of Theorem 30.1 describes an algorithm for interpolation based on solving the set (30.4) of linear equations. Using the LU decomposition algorithms of Chapter 28, we can solve these equations in time $O(n^3)$.

A faster algorithm for n-point interpolation is based on ***Lagrange's formula***:

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \ne k} (x - x_j)}{\prod_{j \ne k} (x_k - x_j)} . \tag{30.5}$$

You may wish to verify that the right-hand side of equation (30.5) is a polynomial of degree-bound n that satisfies $A(x_k) = y_k$ for all k. Exercise 30.1-5 asks you how to compute the coefficients of A using Lagrange's formula in time $\Theta(n^2)$.

Thus, n-point evaluation and interpolation are well-defined inverse operations that transform between the coefficient representation of a polynomial and a point-value representation.[1] The algorithms described above for these problems take time $\Theta(n^2)$.

The point-value representation is quite convenient for many operations on polynomials. For addition, if $C(x) = A(x) + B(x)$, then $C(x_k) = A(x_k) + B(x_k)$ for any point x_k. More precisely, if we have a point-value representation for A,

[1] Interpolation is a notoriously tricky problem from the point of view of numerical stability. Although the approaches described here are mathematically correct, small differences in the inputs or round-off errors during computation can cause large differences in the result.

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\} ,$$

and for B,

$$\{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{n-1}, y'_{n-1})\}$$

(note that A and B are evaluated at the *same* n points), then a point-value representation for C is

$$\{(x_0, y_0 + y'_0), (x_1, y_1 + y'_1), \ldots, (x_{n-1}, y_{n-1} + y'_{n-1})\} .$$

Thus, the time to add two polynomials of degree-bound n in point-value form is $\Theta(n)$.

Similarly, the point-value representation is convenient for multiplying polynomials. If $C(x) = A(x)B(x)$, then $C(x_k) = A(x_k)B(x_k)$ for any point x_k, and we can pointwise multiply a point-value representation for A by a point-value representation for B to obtain a point-value representation for C. We must face the problem, however, that degree(C) = degree(A) + degree(B); if A and B are of degree-bound n, then C is of degree-bound $2n$. A standard point-value representation for A and B consists of n point-value pairs for each polynomial. When we multiply these together, we get n point-value pairs, but we need $2n$ pairs to interpolate a unique polynomial C of degree-bound $2n$. (See Exercise 30.1-4.) We must therefore begin with "extended" point-value representations for A and for B consisting of $2n$ point-value pairs each. Given an extended point-value representation for A,

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{2n-1}, y_{2n-1})\} ,$$

and a corresponding extended point-value representation for B,

$$\{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{2n-1}, y'_{2n-1})\} ,$$

then a point-value representation for C is

$$\{(x_0, y_0 y'_0), (x_1, y_1 y'_1), \ldots, (x_{2n-1}, y_{2n-1} y'_{2n-1})\} .$$

Given two input polynomials in extended point-value form, we see that the time to multiply them to obtain the point-value form of the result is $\Theta(n)$, much less than the time required to multiply polynomials in coefficient form.

Finally, we consider how to evaluate a polynomial given in point-value form at a new point. For this problem, we know of no simpler approach than converting the polynomial to coefficient form first, and then evaluating it at the new point.

Fast multiplication of polynomials in coefficient form

Can we use the linear-time multiplication method for polynomials in point-value form to expedite polynomial multiplication in coefficient form? The answer hinges

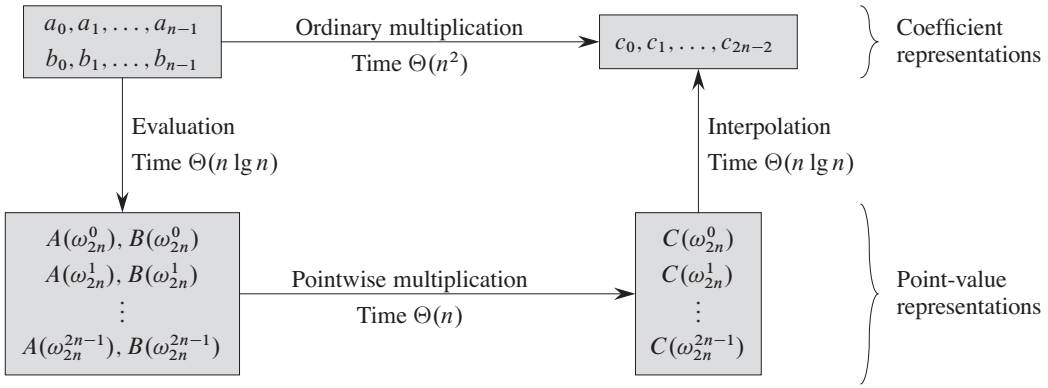

Figure 30.1 A graphical outline of an efficient polynomial-multiplication process. Representations on the top are in coefficient form, while those on the bottom are in point-value form. The arrows from left to right correspond to the multiplication operation. The ω_{2n} terms are complex $(2n)$th roots of unity.

on whether we can convert a polynomial quickly from coefficient form to point-value form (evaluate) and vice versa (interpolate).

We can use any points we want as evaluation points, but by choosing the evaluation points carefully, we can convert between representations in only $\Theta(n \lg n)$ time. As we shall see in Section 30.2, if we choose "complex roots of unity" as the evaluation points, we can produce a point-value representation by taking the discrete Fourier transform (or DFT) of a coefficient vector. We can perform the inverse operation, interpolation, by taking the "inverse DFT" of point-value pairs, yielding a coefficient vector. Section 30.2 will show how the FFT accomplishes the DFT and inverse DFT operations in $\Theta(n \lg n)$ time.

Figure 30.1 shows this strategy graphically. One minor detail concerns degree-bounds. The product of two polynomials of degree-bound n is a polynomial of degree-bound $2n$. Before evaluating the input polynomials A and B, therefore, we first double their degree-bounds to $2n$ by adding n high-order coefficients of 0. Because the vectors have $2n$ elements, we use "complex $(2n)$th roots of unity," which are denoted by the ω_{2n} terms in Figure 30.1.

Given the FFT, we have the following $\Theta(n \lg n)$-time procedure for multiplying two polynomials $A(x)$ and $B(x)$ of degree-bound n, where the input and output representations are in coefficient form. We assume that n is a power of 2; we can always meet this requirement by adding high-order zero coefficients.

1. *Double degree-bound:* Create coefficient representations of $A(x)$ and $B(x)$ as degree-bound $2n$ polynomials by adding n high-order zero coefficients to each.