Combinatorial Optimization: Algorithms and Complexity

CHRISTOS H. PAPADIMITRIOU

Massachusetts Institute of Technology
National Technical University of Athens

KENNETH STEIGLITZ

Princeton University

PRENTICE-HALL, INC.

Englewood Cliffs, New Jersey 07632

Contents

2

The Simplex Algorithm

2.1 Forms of the Linear Programming Problem

The linear programming problem defined in the last chapter was not the most general one; we could have had some inequality as well as equality constraints and we could have had some variables unconstrained in sign, as well as variables restricted to be nonnegative. We now define the most general form of a linear program as follows.

Definition 2.1

Given an $m \times n$ integer matrix A with rows a_i', let M be the set of row indices corresponding to equality constraints, and let $\bar{M}$ be those corresponding to inequality constraints. Similarly, let $x \in R^n$ and let N be the column indices corresponding to constrained variables and $\bar{N}$ those corresponding to unconstrained variables. Then an instance of the *general LP* is defined by

$$\begin{aligned} &\min c'x \\ &a_i'x = b_i \qquad i \in M \\ &a_i'x \geq b_i \qquad i \in \bar{M} \\ &x_j \geq 0 \qquad j \in N \\ &x_j \gtrless 0 \qquad j \in \bar{N} \end{aligned} \tag{2.1}$$

where b is an m-vector of integers and c an n-vector of integers. □

Example 2.1 (Diet Problem)

One of the first problems ever formulated as an LP is the *diet problem* [Sti]. We consider the problem faced by a homemaker when buying food. He has a choice of n foods, and each food has some of each of m nutrients. Suppose

a_{ij} = amount of ith nutrient in a unit of the jth food, $i = 1, \ldots, m$, $j = 1, \ldots, n$

r_i = yearly requirement of ith nutrient, $i = 1, \ldots, m$

x_j = yearly consumption of the jth food, $j = 1, \ldots, n$, in units.

c_j = cost per unit of the jth food, $j = 1, \ldots, n$.

A yearly diet is represented by a choice of a vector $x \geq 0$. That such a diet satisfies the minimal nutritional requirements is expressed by

$$Ax \geq r$$

If we want to find the least expensive diet that is nutritionally adequate, we then need to consider the LP

$$\begin{aligned} &\min c'x \\ &Ax \geq r \\ &x \geq 0 \end{aligned} \tag{2.2}$$

□

The form of LP obtained in the diet problem and the form given in Chapter 1 are common enough to be given special names. We use the following terminology.

Definition 2.2

An LP in the form of (2.2) is said to be in *canonical form*. An LP in the form of (2.3)

$$\begin{aligned} &\min c'x \\ &Ax = b \\ &x \geq 0 \end{aligned} \tag{2.3}$$

is said to be in *standard form*. Finally, an LP in the form of (2.1) is said to be in *general form*. □

We next prove that *the canonical, standard, and general forms are all equivalent*. By this we mean that an instance in one form can be converted to one in another form by a simple transformation, in such a way that the two instances have the same solution. The canonical and standard forms are both already in general form, so we need show only that a general-form problem can be put in canonical and standard forms.

1. To put a general-form problem in canonical form, we need to eliminate any equality constraints and unconstrained variables. Given an equality constraint in the general-form program

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

we can replace this with two inequality constraints

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

and

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

Given an unconstrained variable x_j in the general-form program

$$x_j \gtrless 0$$

we create two variables x_j^+ and x_j^- in the canonical-form program and write

$$x_j = x_j^+ - x_j^- \quad \text{where} \quad x_j^+ \geq 0, \quad x_j^- \geq 0$$

2. To put a general-form problem in standard form, we need to eliminate inequality constraints; unconstrained variables can be eliminated as above. Given an inequality constraint in the general-form program

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

introduce the variable s_i in the canonical problem and write

$$\sum_{j=1}^{n} a_{ij}x_j - s_i = b_i, \qquad s_i \geq 0$$

The variable s_i introduced in this transformation is called a *surplus* variable; it represents the amount by which the left-hand side of the inequality exceeds the right-hand side. If, when formulating an LP, we get an inequality of the form

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

we can introduce a variable s_i and write

$$\sum_{j=1}^{n} a_{ij}x_j + s_i = b_i, \qquad s_i \geq 0$$

Such a variable is called a *slack* variable.

2.2 Basic Feasible Solutions

Our goal now is to develop the simplex algorithm for solving LP's, and it is convenient to assume we are given an LP in *standard form*

$$\begin{aligned} &\min c'x \\ &Ax = b \qquad (A \text{ is an } m \times n \text{ matrix of integers, and } m < n) \\ &\;\; x \geq 0 \end{aligned}$$

which we do without loss of generality by the results in the previous section.

We argued intuitively in Example 1.3 that there should always be an optimal "corner" of the convex feasible set F of an LP. There are two ways to define such "corners" precisely—one geometric, and one algebraic. For our algebraic definition we need the following assumption, which we shall see later is hardly restrictive:

Assumption 2.1 There are m linearly independent columns A_j of A. That is, A is of rank m.

Definition 2.3

A *basis* of A is a linearly independent collection $\mathcal{B} = \{A_{j_1}, \ldots, A_{j_m}\}$. We can alternatively think of $\mathcal{B}$ as an $m \times m$ nonsingular matrix $B = [A_{j_i}]$. The *basic solution* corresponding to $\mathcal{B}$ is a vector $x \in R^n$ with

$$\begin{aligned} &x_j = 0 \text{ for } A_j \notin \mathcal{B} \\ &x_{j_k} = \text{the } k\text{th component of } B^{-1}b, \qquad k = 1, \ldots, m. \quad \square \end{aligned}$$

Thus a basic solution x can be found by the following procedure:

1. Choose a set $\mathcal{B}$ of linearly independent columns of A.
2. Set all components of x corresponding to columns not in $\mathcal{B}$ equal to zero.
3. Solve the m resulting equations to determine the remaining components of x. These are the *basic variables.*

Example 2.2

Let us consider the LP

$$
\begin{array}{lllllllll}
\min & 2x_2 + x_4 + 5x_7 & & & & & & & \\
x_1 + & x_2 + x_3 + x_4 & & & & & & = 4 \\
x_1 & & & + x_5 & & & & = 2 \\
 & & x_3 & & + x_6 & & & = 3 \\
 & 3x_2 + x_3 & & & & + & x_7 & = 6 \\
x_1, & x_2, \quad x_3, & x_4, & x_5, & x_6, & & x_7 & \geq 0
\end{array}
$$

One basis is certainly $\mathcal{B} = \{A_4, A_5, A_6, A_7\}$, which corresponds to the matrix $B = I$. The corresponding basic solution is $x = (0, 0, 0, 4, 2, 3, 6)$. Another basis is $\mathcal{B}' = \{A_2, A_5, A_6, A_7\}$, with basic solution $x' = (0, 4, 0, 0, 2, 3, -6)$. Notice that x' is *not* a feasible solution, since $x'_7 < 0$. □

One can bound from above the absolute value of the components of any basic solution by using our assumption that the entries of A, b, and c are integers.

Lemma 2.1 *Let $x = (x_1, \ldots, x_n)$ be a basic solution. Then*

$$|x_j| \leq m!\, \alpha^{m-1} \beta$$

where

$$\alpha = \max_{i,j} \{|a_{ij}|\}$$

and

$$\beta = \max_{j=1,\ldots,m} \{|b_j|\}$$

Proof This is true for x_j not a basic variable, because then $x_j = 0$. For basic variables, recall that x_j is the sum of m products of elements of B^{-1} by elements of b. Now, each element of B^{-1} is, by definition of the inverse, equal to an $(m-1) \times (m-1)$ determinant divided by a nonzero $m \times m$ determinant. By integrality, the denominator is of absolute value at least 1. The determinant of the numerator is the sum of $(m-1)!$ products of $m-1$ elements of A; therefore it has absolute value no greater than $(m-1)!\, \alpha^{m-1}$. Because each x_j is the sum of m elements of B^{-1} multiplied by an element of b, we have

$$|x_j| \leq m!\, \alpha^{m-1} \beta. \qquad \square$$

This bound will be used many times in future arguments.

Definition 2.4

If a basic solution x is in F, then x is a *basic feasible solution* (bfs). □

For example, in the LP of Example 2.2, $x = (0, 0, 0, 4, 2, 3, 6)$ is a bfs. Basic feasible solutions play a very central role in both the theory and the computing practice of LP. One aspect of their importance is expressed in the following lemma, stating that *all* basic feasible solutions are potential uniquely optimal solutions of the corresponding LP.

Lemma 2.2 *Let x be a bfs of*

$$\begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned}$$

corresponding to the basis $\mathcal{B}$. Then there exists a cost vector c such that x is the unique optimal solution of the LP

$$\begin{aligned} &\min c'x \\ &Ax = b \\ &x \geq 0 \end{aligned}$$

Proof Consider the cost vector c defined by

$$c_j = \begin{cases} 0 & \text{if } A_j \in \mathcal{B} \\ 1 & \text{if } A_j \notin \mathcal{B} \end{cases}$$

The cost of the bfs x is $c'x = 0$; obviously, x is optimal because all c_j's are nonnegative. Furthermore, if any other feasible solution y is also going to have zero cost, it must be the case that $y_j = 0$ for all $A_j \notin \mathcal{B}$. Hence y must be equal to x, and x is uniquely optimal. □

It is not at all certain, however, that all LP's have bfs's. For example, if $F = \varnothing$, naturally there can be no bfs. It is convenient, however, to exclude this pathological case at this point. We shall come back, in time, to see how one can remove this assumption.

Assumption 2.2 The set F of feasible points is not empty.

We can now show that bfs's do exist.

Theorem 2.1 *Under Assumptions 2.1 and 2.2, at least one bfs exists.*

Proof Assume that F contains a solution x with $t > m$ nonzero components, and in fact that x is the solution in F with the largest number of zero components. Without loss of generality, we have

$$x_1, \ldots, x_t > 0; \qquad x_{t+1}, \ldots, x_n = 0$$

Consider the first t columns of A. They obviously satisfy

$$A_1 x_1 + \cdots + A_t x_t = b \tag{2.4}$$

Let r be the rank of the matrix of these t columns; $r > 0$, because if $r = 0$ the bfs $x = 0$ is in F. Also, $r \leq m < t$. We may thus assume that the matrix

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1r} \\ a_{21} & a_{22} & \cdots & a_{2r} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ a_{r1} & a_{r2} & \cdots & a_{rr} \end{bmatrix}$$

is nonsingular. Therefore we can solve Eq. 2.4 to express $x_1, \ldots, x_r$ in terms of $x_{r+1}, \ldots, x_t$. In other words

$$x_j = \beta_j + \sum_{i=r+1}^{t} \alpha_{ij} x_i, \qquad j = 1, \ldots, r$$

Now, let θ be the quantity

$$\theta = \min \{x_{r+1}, \theta_1\}$$

where

$$\theta_1 = \min_{\alpha_{r+1,i} > 0} \left\{ \frac{x_i}{\alpha_{r+1,i}}, i = 1, \ldots, r \right\}$$

Construct a new feasible solution $\hat{x}$ by

$$\hat{x}_j = \begin{cases} x_j - \theta & \text{if } j = r + 1 \\ x_j & \text{if } j > r + 1 \\ \beta_j + \sum_{i=r+1}^{t} \alpha_{ij} \hat{x}_{ij} & \text{if } j < r + 1 \end{cases}$$

Then, for $j \leq r$, $\hat{x}_j = x_j - \alpha_{r+1,j}\theta$. If $\theta = x_{r+1}$, then $\hat{x}_{r+1} = 0$; if $\theta = \theta_1 = x_k/\alpha_{r+1,k}$ for some $k \leq r$, then $\hat{x}_k = 0$. In either case, $\hat{x}$ is a feasible solution with one more zero component than x, which is a contradiction.

This argument shows that there is a solution x with $t \leq m$ nonzero components, and furthermore that the corresponding columns can be assumed to be linearly independent. This set of columns can then be augmented to a basis for x because A is of rank m. □

One final convenient assumption, which we shall also show to be unnecessary later: We shall assume that the LP has a finite minimum value of the objective function $c'x$.

Assumption 2.3 The set of real numbers $\{c'x : x \in F\}$ is bounded from below.

Even though the cost $c'x$ of an LP is bounded from below, the feasible set may still extend infinitely far in some directions. We conclude this section by showing that, under Assumption 2.3, we can nevertheless restrict our attention to LP's with *bounded* feasible sets F. More precisely, F can be assumed to lie within a suitably large hypercube.

Theorem 2.2 *Let Assumptions 2.1 through 2.3 hold for the LP*

$$\begin{aligned} &\min c'x \\ &Ax = b \\ &x \geq 0 \end{aligned} \tag{LP}$$

Then LP is equivalent, in the sense that it has the same optimal value of its cost function:*

$$\begin{aligned} &\min c'x \\ &Ax = b \\ &x \geq 0 \\ &x \leq M \end{aligned} \tag{LP*}$$

where

$$\begin{aligned} M &= (m+1)!\,\alpha^m \beta \\ \alpha &= \max \{|a_{ij}|, |c_j|\} \\ \beta &= \max \{|b_i|, |z|\} \end{aligned}$$

and z is the greatest lower bound of the set $\{c'x\colon Ax = b, x \geq 0\}$.

Proof Consider the set of real numbers

$$G = \{c'x\colon Ax = b, x \geq 0\}$$

It is not hard to show that G is closed (see Problem 2.10). Therefore there is a point x that is feasible in LP and achieves the greatest lower bound z. Next consider the set of points that satisfy the constraints

$$\begin{aligned} c'x &= z \\ Ax &= b \\ x &\geq 0 \end{aligned} \tag{2.5}$$

This set is then nonempty, and in fact consists of all the *optimal* feasible solutions to LP.

Assume first that the equations in (2.5) are of rank $m + 1$. Then Theorem 2.1 implies that (2.5) has a bfs, and Lemma 2.1 shows that its components satisfy the desired bound. Hence the constraints $x \leq M$ do not change the optimal solution of LP.

We have left to consider the case in which the equations in (2.5) are of rank m. In that case c' can be written as a linear combination $\sum d_i a_i'$ of the rows of A, and the cost $c'x = \sum d_i b_i$ is a constant for all feasible points of LP. Therefore LP has an optimal bfs, and its components by Lemma 2.1 are bounded by a number no larger than M. □

From now on we shall use this result to assume that F is *always bounded.*

2.3
The Geometry of Linear Programs

We shall now give some important definitions and results pertaining to an alternative *geometric* way of viewing LP.

2.3.1 Linear and Affine Spaces

Consider the vector space R^d. A (*linear*) *subspace* S of R^d is a subset of R^d closed under vector addition and scalar multiplication. Equivalently, a subspace S of R^d is the set of points in R^d that satisfy a set of homogenous linear equations:

$$S = \{x \in R^d : a_{j1}x_1 + \cdots + a_{jd}x_d = 0, \quad j = 1, \ldots, m\} \tag{2.6}$$

It is well known that every subspace S has a *dimension*, $\dim(S)$, equal to the maximum number of linearly independent vectors in it. Equivalently, $\dim(S) = d - \operatorname{rank}([a_{ij}])$, where $[a_{ij}]$ is the matrix of the coefficients in (2.6) above.

An *affine subspace* A of R^d is a linear subspace S translated by a vector u: $A = \{u + x : x \in S\}$. The *dimension* of A is that of S. Equivalently, an affine subspace A of R^d is the set of all points satisfying a set of (inhomogeneous) equations

$$A = \{x \in R^d : a_{j1}x_1 + \cdots + a_{jd}x_d = b_j; \quad j = 1, \ldots, m\}$$

The dimension of *any subset* of R^d is the smallest dimension of any affine subspace which contains it. For example, any line segment has dimension 1; any set of k points, $k \leq d + 1$, has dimension at most $k - 1$. The dimension of the set F defined by the LP (satisfying Assumptions 2.1 and 2.2)

$$\begin{aligned} &\min c'x \\ &Ax = b \qquad A \text{ an } m \times d \text{ matrix} \\ &x \geq 0 \end{aligned}$$

is therefore at most $d - m$.

2.3.2 Convex Polytopes

An affine subspace of R^d of dimension $d - 1$ is called a *hyperplane*. Alternatively, a hyperplane is a set of points x satisfying

$$a_1x_1 + a_2x_2 + \cdots + a_dx_d = b$$

with not all a's equal to zero. A hyperplane defines two (*closed*) *halfspaces*, namely the sets of points satisfying, respectively,

$$a_1x_1 + \cdots + a_dx_d \geq b$$
$$a_1x_1 + \cdots + a_dx_d \leq b$$

A halfspace is a convex set. Therefore the intersection of halfspaces is also convex. The intersection of a finite number of halfspaces, when it is bounded and nonempty, is called a *convex polytope*, or simply a *polytope*.

We shall henceforth be interested only in convex polytopes that are included in the nonnegative orthant; in other words, by *convention*, d of the halfspaces defining a polytope will always be $x_j \geq 0$, $j = 1, \ldots, d$.

Example 2.3

The 3-dimensional polytope P of Figure 2–1 is the intersection of the halfspaces indicated by the inequalities in (2.7). As required, P is bounded, because it can easily be shown to be totally contained in the cube $0 \leq x_1, x_2, x_3 \leq 3$.

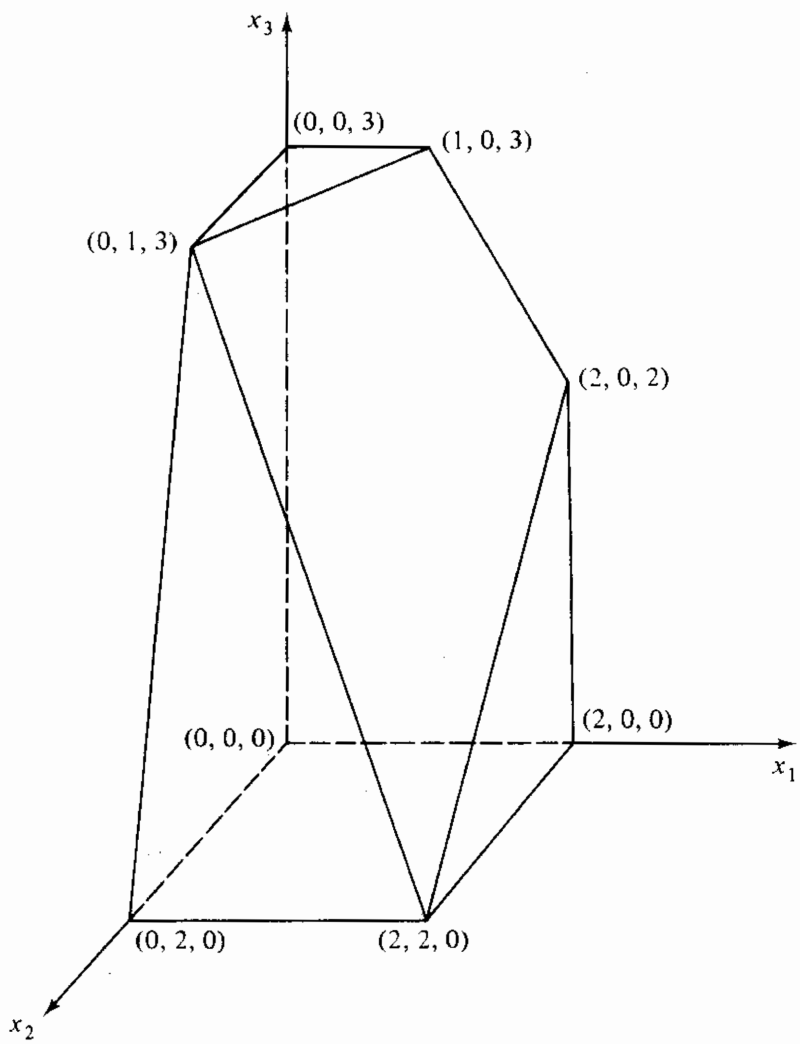


Figure 2–1 The 3-dimensional polytope in Example 2.3.

$$\begin{array}{rcrcrcl} x_1 & + & x_2 & + & x_3 & \le & 4 \\ x_1 & & & & & \le & 2 \\ & & & & x_3 & \le & 3 \\ & & 3x_2 & + & x_3 & \le & 6 \\ x_1 & & & & & \ge & 0 \\ & & x_2 & & & \ge & 0 \\ & & & & x_3 & \ge & 0 \quad \square \end{array} \tag{2.7}$$

Let P be a convex polytope of dimension d and let HS be a halfspace defined by hyperplane H. If the intersection $f = P \cap HS$ is a subset of H—in other words, P and HS just "touch in their exteriors"—then f is called a *face* of P and H is the *supporting hyperplane defining* f. We have three distinguished kinds of faces.

A *facet* is a face of dimension $d - 1$.

A *vertex* is a face of dimension zero (a point).

An *edge* is a face of dimension one (a line segment).

Example 2.3 (Continued)

Figure 2–2 shows the polytope P together with three hyperplanes H_1, H_2, and H_3, which define three faces: a facet, an edge and a vertex, respectively. $\square$

The following are fairly intuitive observations, which can easily be proved rigorously [Gru, Roc, YG]. The hyperplane defining a facet corresponds to a defining halfspace of the polytope. The converse is not always true: If we add the halfspace $x_2 \le 2$ to those defining P, P would remain the same. However, the new halfspace would not define a facet—it would, however, define an edge, the line segment $[(0, 2, 0), (2, 2, 0)]$. The reason is that, intuitively, $x_2 \le 2$ is *redundant* in defining P.

A vertex is the "corner" of the polytope that we alluded to earlier with less precision. An edge is always a line segment joining two vertices. Not every pair of vertices defines an edge, though: The segment $[(0, 0, 3), (2, 2, 0)]$ is *not* an edge; neither is $[(1, 0, 3), (2, 2, 0)]$ an edge.

Another fairly intuitive fact, though harder to prove, is that every point in P is the convex combination of its vertices—in fact, it can be shown that four vertices ($d + 1$ for dimension d) always suffice. For example, the point $(1, 1, 1)$, which is in the interior of P, can be rewritten as $(1, 1, 1) = \frac{1}{2}(2, 2, 0) + \frac{1}{3}(0, 0, 3) + \frac{1}{6}(0, 0, 0)$. We now state a general theorem to this effect.

(2.7)

ɔe a halfspace
ıbset of H—in
s called a *face*
e distinguished

:planes H_1, H_2,
:x, respectively.

:asily be proved
orresponds to a
true: If we add
same. However,
'ever, define an
ntuitively, $x_2 \leq$

l to earlier with
rtices. Not every
), (2, 2, 0)] is *not*

that every point
n be shown that
ɪr example, the
ən as (1, 1, 1) =
theorem to this

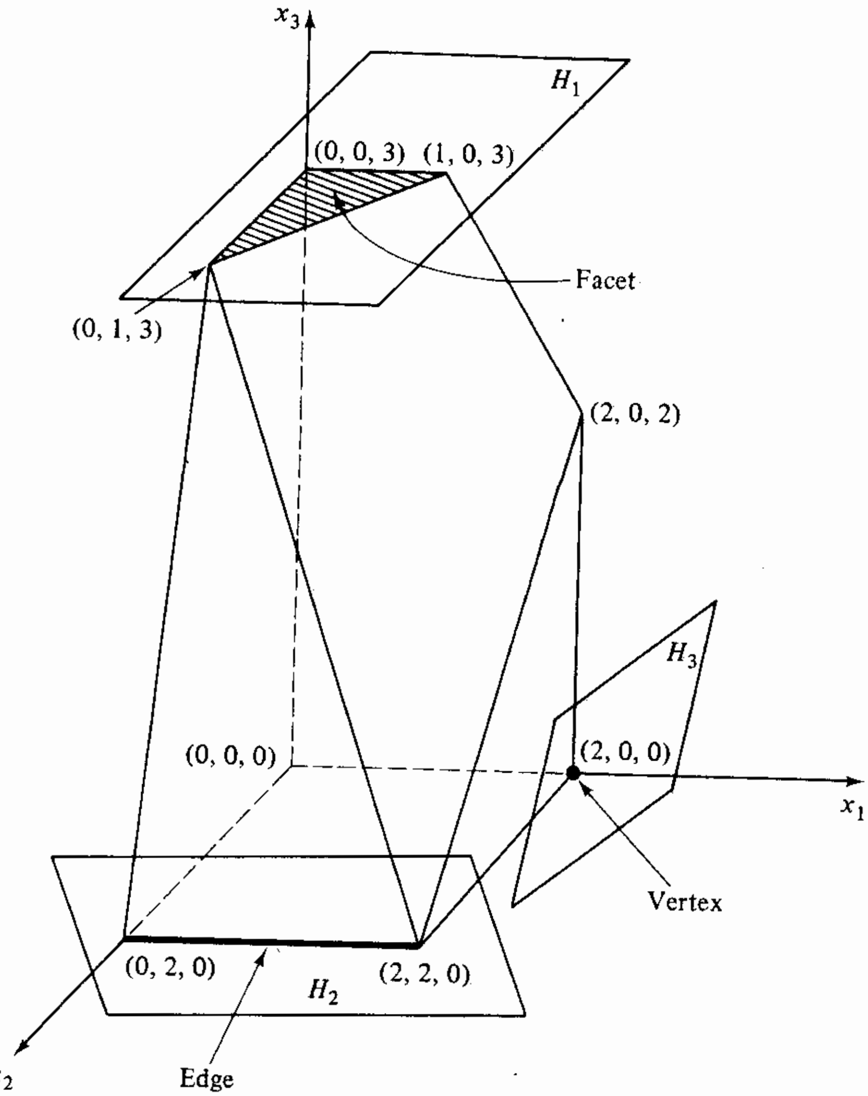


Figure 2–2

Theorem 2.3 **[Gru, Roc, YG]**

(*a*) *Every convex polytope is the convex hull of its vertices.*

(*b*) *Conversely, if V is a finite set of points, then the convex hull of V is a convex polytope P. The set of vertices of P is a subset of V.*

2.3.3 Polytopes and LP

By Theorem 2.3, a polytope P can be thought of in several different ways.

1. *As the convex hull of a finite set of points.* This point of view is fairly convenient when we are given only the vertices of the polytope. This

will be the case in Chapters 13 and 19 in connection with certain combinatorial problems.

2. *As the intersection of many halfspaces, as long as this intersection is bounded.* This is a natural way to look at a polytope when these inequalities are explicitly given. We shall next see that LP is such a situation.

3. A third aspect of a polytope is an algebraic version of 2. above. Let

$$\begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned} \tag{2.8}$$

be the defining equations and inequalities of the feasible region F of an LP satisfying Assumptions 2.1, 2.2, and 2.3. Since $\operatorname{rank}(A) = m$, where A is an $m \times n$ matrix, we can assume that the equations $Ax = b$ are of the form

$$x_i = b_i - \sum_{j=1}^{n-m} a_{ij}x_j, \qquad i = n - m + 1, \ldots, n \tag{2.8$'$}$$

because otherwise we can find a basis B of A (without loss of generality the last m rows of A) and premultiply (2.8) by B^{-1} to obtain (2.8$'$). Thus (2.8) is equivalent to the inequalities

$$\begin{aligned} b_i - \sum_{j=1}^{n-m} a_{ij}x_j \geq 0 &\qquad i = n - m + 1, \cdots, n \\ x_j \geq 0 &\qquad j = 1, \ldots, n - m \end{aligned} \tag{2.9}$$

However, (2.9) describes the intersection of n halfspaces, which by Theorem 2.2 is bounded. Hence (2.9) defines a *convex polytope* $P \subseteq R^{n-m}$.

Conversely, let P be a polytope in R^{n-m}. The n halfspaces defining P can be expressed by the inequalities

$$h_{i1}x_1 + \cdots + h_{i,n-m}x_{n-m} + g_i \leq 0 \qquad i = 1, \ldots, n \tag{2.10}$$

By our *convention*, we may assume that the first $n - m$ inequalities in (2.10) are of the form

$$x_i \geq 0 \qquad i = 1, \ldots, n - m$$

Let H be the matrix of the coefficients of the remaining inequalities. We can introduce m slack variables $x_{n-m+1}, \ldots, x_n$ to obtain

$$\begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned}$$

where the $m \times n$ matrix $A = [H|I]$ and $x \in R^n$. Thus any polytope (satisfying our convention) can be alternatively viewed as the feasible region F of an LP via a simple transformation. Furthermore, any point $\hat{x} = (x_1, \ldots, x_{n-m}) \in P$ can be transformed to $x = (x_1, \ldots, x_n) \in F$ by defining

$$x_i = -g_i - \sum_{j=1}^{n-m} h_{ij} x_j \qquad i = n - m + 1, \ldots, n \tag{2.11}$$

Conversely, any $x = (x_1, \ldots, x_n) \in F$ can be easily transformed to $\hat{x} = (x_1, \ldots, x_{n-m}) \in P$ by simply truncating the last m coordinates of x.

We can now show how these three points of view affect our notion of a "corner."

Theorem 2.4 *Let P be a convex polytope, $F = \{x\colon Ax = b,\ x \geq 0\}$ the corresponding feasible set of an LP, and $\hat{x} = (x_1, \ldots, x_{n-m}) \in P$. Then the following are equivalent.*

(*a*) *The point $\hat{x}$ is a vertex of P.*

(*b*) *If $\hat{x} = \lambda\hat{x}' + (1 - \lambda)\hat{x}''$, with $\hat{x}', \hat{x}'' \in P$, $0 < \lambda < 1$, then $\hat{x}' = \hat{x}'' = \hat{x}$ (in other words, $\hat{x}$ cannot be the* strict *convex combination of points of P).*

(*c*) *The corresponding vector x in F defined by (2.11) is a bfs of F.*

Proof (a) $\Rightarrow$ (b) Suppose that $\hat{x}$ is a vertex and yet there are points $\hat{x}'$, $\hat{x}'' \in P$ different from $\hat{x}$ such that, for $0 < \lambda < 1$, $\hat{x} = \lambda\hat{x}' + (1 - \lambda)\hat{x}''$. Since $\hat{x}$ is a vertex, there is a halfspace $HS = \{\hat{x} \in R^{n-m}\colon h'\hat{x} \leq g\}$ such that $HS \cap P = \{\hat{x}\}$. Thus $\hat{x}', \hat{x}'' \notin HS$, and hence $h'\hat{x}' > g$ and $h'\hat{x}'' > g$. It follows that $h'\hat{x} = h'(\lambda\hat{x}' + (1 - \lambda)\hat{x}'') > g$ and $\hat{x} \notin HS$, a contradiction.

(b) $\Rightarrow$ (c) Suppose that $\hat{x}$ has Property (b), and consider the corresponding element x of F. Consider the subset $\mathcal{B}$ of the columns of A defined by $\mathcal{B} = \{A_j\colon x_j > 0,\ 1 \leq j \leq n\}$. We wish first to show that this is a linearly independent set of columns. Suppose it is not. Then there are integers d_j, not all 0, such that

$$\sum_{A_j \in \mathcal{B}} d_j A_j = 0 \tag{2.12}$$

Since $x \in F$ we have

$$\sum_{A_j \in \mathcal{B}} x_j A_j = b, \tag{2.13}$$

and also

$$x_j \geq 0 \qquad j = 1, \ldots, n$$

Now multiply (2.12) by some number θ and add and subtract from (2.13).

$$\sum_{A_j \in \mathcal{B}} (x_j \pm \theta\, d_j) A_j = b$$

Since $x_j > 0$ for $A_j \in \mathcal{B}$, we can choose a positive and sufficiently small θ such that

$$x_j \pm \theta\, d_j \geq 0 \text{ for all } A_j \in \mathcal{B}$$

Thus we have found two points defined by

$$x'_j = \begin{cases} x_j - \theta\, d_j & A_j \in \mathcal{B} \\ 0 & A_j \notin \mathcal{B} \end{cases}$$

and

$$x''_j = \begin{cases} x_j + \theta\, d_j & A_j \in \mathcal{B} \\ 0 & A_j \notin \mathcal{B} \end{cases}$$

such that $x', x'' \in F$ or $\hat{x}', \hat{x}'' \in P$, and yet $\hat{x} = \frac{1}{2}\hat{x}' + \frac{1}{2}\hat{x}''$, a contradiction.

We have shown that the set of columns $\mathcal{B}$ is linearly independent, and so $|\mathcal{B}| \leq m$. Since we have assumed that there are m linearly independent columns of A, we can always augment the set $\mathcal{B}$ so that it is linearly independent and has m vectors. These then form basic columns, which render x a bfs.

(c) $\Rightarrow$ (a) If $y = (y_1, \ldots y_n)$ is a bfs of $Ax = b, x \geq 0$, then, by Lemma 2.2, there exists a cost vector c such that y is the unique vector $x \in R^n$ satisfying

$$c'x \leq c'y$$
$$Ax = b$$
$$x \geq 0$$

It is easy to see, however, that this means that $\hat{y} = (y_1, \ldots, y_{n-m})$ is the unique point in R^{n-m} satisfying

$$d'\hat{x} \leq d'\hat{y} \qquad \hat{x} \in P$$

where

$$d_i = c_i - \sum_{j=1}^{m} h_{n-m+j,i}c_{n-m+j} \qquad i = 1, \ldots, n-m$$

Hence $\hat{y}$ is indeed a vertex of P, with supporting hyperplane defined by $d'\hat{x} = d'\hat{y}$. □

In Sec. 2.9 we shall derive a very similar characterization of the *edges* of a polytope P.

By Theorem 2.4 we have a correspondence between vertices of P and bases of A. Given two different vertices of P, u and u', the corresponding bases $\mathcal{B}$ and $\mathcal{B}'$ must be different, because a basis uniquely determines a bfs and hence a vertex. However, *two different bases $\mathcal{B}$ and $\mathcal{B}'$ may correspond to the same bfs x.*

Example 2.4

Recall the LP and polytope of Figure 2–1. The matrix A is

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 3 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} 4 \\ 2 \\ 3 \\ 6 \end{bmatrix}$$

Consider the bases $\mathcal{B} = \{A_1, A_2, A_3, A_6\}$ and $\mathcal{B}' = \{A_1, A_2, A_4, A_6\}$. Both have

$$B^{-1}b = B'^{-1}b = (2, 2, 0, 0, 0, 3, 0)$$

A look at Figure 2–1 explains what has happened. To calculate the vertex corresponding to $\mathcal{B}$, we first set $x_4 = x_5 = x_7 = 0$, which means that the corresponding three inequalities must be satisfied by equality, determining the vertex (2, 2, 0) by the intersection of three facets. Now in $\mathcal{B}'$ we replace the constraint $x_1 + x_2 + x_3 \leq 4$ by $x_3 \geq 0$. But $x_3 = 0$ also happens to pass through the same vertex (2, 2, 0), and so nothing has changed. Thus a vertex like this must lie on more than $n - m = 3$ facets; equivalently, the bfs must have more than $n - m = 3$ zeros. □

Definition 2.5

A bfs (and the corresponding vertex) is called *degenerate* if it contains more than $n - m$ zeros. □

We now give the essential result of the above discussion.

Theorem 2.5 *If two distinct bases correspond to the same bfs x, then x is degenerate.*

Proof Suppose that $\mathcal{B}$ and $\mathcal{B}'$ both determine the same bfs x. Then x has zeros in the $n - m$ columns not in $\mathcal{B}$; it also must have zeros in the columns in $\mathcal{B} - \mathcal{B}' \neq \varnothing$. Hence it is degenerate. □

We can now show the following, which is tantamount to showing that LP can be solved in a finite number of steps.

Theorem 2.6 *There is an optimal bfs in any instance of LP. Furthermore, if q bfs's are optimal, their convex combinations are also optimal.*

Proof By Theorem 2.4 and its proof, this is equivalent to proving that there is an optimal vertex of P and that if q vertices of P are optimal, their convex combinations are also, where the linear cost is $d'x$. The set P is closed and bounded, so the linear function d achieves its minimum in P. Let x_0 be an optimal solution and let $x_1, \ldots, x_N$ be the vertices of P. We know from Theorem 2.3 that x_0 can be written

$$x_0 = \sum_{i=1}^{N} \alpha_i x_i$$

where

$$\sum_{i=1}^{N} \alpha_i = 1, \qquad \alpha_i \geq 0$$

Let j be the index corresponding to the vertex with lowest cost. Then

$$d'x_0 = \sum_{i=1}^{N} \alpha_i d'x_i \geq d'x_j \sum_{i=1}^{N} \alpha_i = d'x_j$$

which shows that x_j is optimal.

For the second part of the result, assume that vertices $x_{j_1}, \ldots, x_{j_q}$ are optimal, and let y be a convex combination of these vertices. Then y is optimal, because

$$d'y = d' \sum_{i=1}^{q} \alpha_i x_{j_i} = \sum_{i=1}^{q} \alpha_i (d'x_{j_i}) = d'x_{j_1} \qquad \square$$

We have thus established that an instance of LP can be solved in a finite number of steps: We need examine the cost only at each vertex of the polytope P. Furthermore, all vertices of P (in fact, all bfs's) can be generated systematically by taking each set of m columns, inverting the corresponding matrix B, and rejecting those that have a negative component of $B^{-1}b$. This is hardly a practical algorithm in a reasonably sized instance, however, since there are just too many possible vertices. With the geometric picture we have of the polytope P and its vertices, we are now in a position to develop the simplex algorithm, in which we move from vertex to vertex in a systematic way, thus avoiding an enumeration of all vertices.

2.4 Moving from bfs to bfs

Let x_0 be a bfs of an instance of LP with matrix A, corresponding to the ordered set of indices of basic columns

$$\mathcal{B} = \{A_{B(i)} : i = 1, \ldots, m\}$$

If the basic components of x_0 are x_{i0}, $i = 1, \ldots, m$, then

$$\sum_{i=1}^{m} x_{i0} A_{B(i)} = b, \quad \text{where} \quad x_{i0} \geq 0 \tag{2.14}$$

where as usual we use $A_j \in R^m$ to represent the jth column of A. The set of basic column vectors $\mathcal{B}$ is linearly independent, by definition, so we can write any nonbasic column $A_j \in R^m$, $A_j \notin \mathcal{B}$ as a linear combination of the basic columns as follows:

$$\sum_{i=1}^{m} x_{ij} A_{B(i)} = A_j \tag{2.15}$$

If we now multiply Eq. 2.15 by a scalar $\theta > 0$ and subtract from Eq. 2.14, we get a most important equation:

$$\sum_{i=1}^{m} (x_{i0} - \theta x_{ij}) A_{B(i)} + \theta A_j = b \tag{2.16}$$

Assume for the moment that x_0 is nondegenerate; then all the $x_{i0} > 0$, and as we increase θ from zero, we move from the bfs to feasible solutions with $m + 1$ strictly positive components. How far can we move θ and still remain feasible? Until the first component $(x_{i0} - \theta x_{ij})$ becomes zero, which occurs at the value

$$\theta_0 = \min_{\substack{i \\ \text{such that} \\ x_{ij} > 0}} \frac{x_{i0}}{x_{ij}} \tag{2.17}$$

Example 2.5

Consider the LP with the constraints of Example 2.2 (or 2.4). The basis $\mathcal{B} = \{A_1, A_3, A_6, A_7\}$ given by $B(1) = 1$, $B(2) = 3$, $B(3) = 6$, $B(4) = 7$ has bfs $x = (2, 0, 2, 0, 0, 1, 4)$. We can write the nonbasic column $A_5 = \text{col}\,(0, 1, 0, 0)$ as

$$\begin{aligned} A_5 &= x_{15}A_1 + x_{25}A_3 + x_{35}A_6 + x_{45}A_7 \\ &= 1 \cdot A_1 - 1 \cdot A_3 + 1 \cdot A_6 + 1 \cdot A_7 \end{aligned}$$

Then Eq. 2.16 becomes

$$(2 - \theta)A_1 + (2 + \theta)A_3 + (1 - \theta)A_6 + (4 - \theta)A_7 + \theta A_5 = b$$

A look at Fig. 2.1 shows that this family of feasible points

$$(2 - \theta, 0, 2 + \theta, 0, \theta, 1 - \theta, 4 - \theta)$$

moves from the vertex (2, 0, 2)—and the bfs (2, 0, 2, 0, 0, 1, 4)—to the vertex (1, 0, 3) and the bfs (1, 0, 3, 0, 1, 0, 3) as θ increases from zero to 1. Equation 2.17 yields $\theta_0 = 1$, and the new basis becomes $\mathcal{B}'$ with $B'(1) = 1$, $B'(2) = 3$, $B'(3) = 5$ and $B'(4) = 7$. □

We now take up two special conditions that might prevail at the bfs x_0.

Special Case 1 If x_0 is degenerate because some $x_{i0} = 0$ and the corresponding x_{ij} is positive, then $\theta_0 = 0$ by (2.17), and we do not move in R^n. We stay at the same vertex, but can think of ourselves as moving to the *new basis* with column j replacing column $B(i)$. We sometimes say in such a case that variable x_j has entered the basis *at zero level.*

Special Case 2 If all the x_{ij}, $i = 1, \ldots, m$ are nonpositive, we can move arbitrarily far without becoming infeasible. In such a case F is unbounded, violating our taking F bounded after Theorem 2.2.

It remains to show that the new point reached by the above process is in fact a bfs.

Theorem 2.7 *Given a bfs x_0 with basic components x_{i0}, $i = 1, \ldots, m$ and basis $\mathcal{B} = \{A_{B(i)}: i = 1, \ldots, m\}$, let j be such that $A_j \notin \mathcal{B}$. Then the new* *feasible solution determined by*

$$\theta_0 = \min_{\substack{i \\ \text{such that} \\ x_{ij} > 0}} \frac{x_{i0}}{x_{ij}} = \frac{x_{l0}}{x_{lj}} \tag{2.18}$$

$$x'_{i0} = \begin{cases} x_{i0} - \theta_0 x_{ij} & i \neq l \\ \theta_0 & i = l \end{cases} \tag{2.19}$$

is a bfs with basis $\mathcal{B}'$ *defined by*

$$B'(i) = \begin{cases} B(i) & i \neq l \\ j & i = l \end{cases} \tag{2.20}$$

When there is a tie in the min operation of (2.18), the new bfs is degenerate.

Proof We need to show that x'_0 with components given by Eq. 2.19 is basic, since it is a feasible solution by the discussion surrounding Eqs. 2.16 and 2.17. Thus, we must show that the set of basic columns $\mathcal{B}'$ is linearly independent.

Suppose then that for some constants d_i we have

$$\sum_{i=1}^{m} d_i A_{B'(i)} = d_l A_j + \sum_{\substack{i=1 \\ i \neq l}}^{m} d_i A_{B(i)} = 0 \tag{2.21}$$

Substituting

$$A_j = \sum_{i=1}^{m} x_{ij} A_{B(i)} \tag{2.22}$$

this becomes

$$\sum_{\substack{i=1 \\ i \neq l}}^{m} (d_l x_{ij} + d_i) A_{B(i)} + d_l x_{lj} A_{B(l)} = 0 \tag{2.23}$$

This is a linear combination of the original basis vectors, so all the coefficients must be zero; in particular $d_l x_{lj} = 0$, and hence $d_l = 0$. Equation 2.21 then implies that the remaining d_i are zero and hence that the new basis is in fact linearly independent.

We conclude the proof by noting that if a tie occurs in the min operation of Eq. 2.18, the corresponding entries in x'_0 become zero by Eq. 2.19, which means x'_0 is degenerate. □

This method of moving from one bfs to another is called *pivoting;* we say column $B(l)$ *leaves* the basis and column j *enters* the basis.

2.5 Organization of a Tableau

In the last section we assumed that we always had available to us the representation of any nonbasic Column A_j in terms of the basic columns, as in Eq. 2.22. It is crucial that we have the coefficients x_{ij} at our fingertips if we are to pursue an algorithm that does a great deal of moving from vertex to vertex. We do this by keeping our set of equations diagonalized with respect to the basic variables.

Suppose that at any stage we keep an $m \times (n+1)$ array of numbers that represents the information in the original equality constraints $Ax = b$. Thus we represent the equations

$$\begin{aligned} 3x_1 + 2x_2 + x_3 \qquad &= 1 \\ 5x_1 + \ \ x_2 + x_3 + x_4 &= 3 \\ 2x_1 + 5x_2 + x_3 + x_5 &= 4 \end{aligned}$$

by

	x_1	x_2	x_3	x_4	x_5
1	3	2	1	0	0
3	5	1	1	1	0
4	2	5	1	0	1

separating the right-hand sides of the equations with a vertical bar and considering them as Column 0. We can multiply a row by a nonzero constant or add a multiple of any row to any other without changing the information in these equations; these are usually called *elementary row operations*. If we have a basis $\mathcal{B}$ available, we can perform elementary row operations until the basic columns form an identity submatrix:

$$A_{B(i)} = e_i = \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \longleftarrow i\text{th row} \\ \\ \\ \\ \\ \end{matrix} \qquad i = 1, \ldots, m$$

where we conventionally use e_i to represent the m-vector with a 1 in the ith row and zero elsewhere. Thus, in our example, if $\mathcal{B} = \{A_3, A_4, A_5\}$, we can multiply Row 1 by -1 and add it to Rows 2 and 3, yielding

	x_1	x_2	x_3	x_4	x_5
1	3	2	1	0	0
2	2	-1	0	1	0
3	-1	3	0	0	1

(2.24)

Column 0 now gives the values of the basic variables $x_{B(i)} = x_{i0}$, $i = 1, \ldots, m$ obtained by setting the nonbasic variables to zero. Notice also that the nonbasic columns contain precisely the numbers x_{ij}; for example,

$$A_1 = 3A_3 + 2A_4 - A_5$$
$$= \sum_{i=1}^{m} x_{i1} A_{B(i)}$$

The calculations necessary to change the basis can therefore be carried out immediately. Suppose, for example, that we wish to bring Column $j = 1$ into the basis; then by Eq. 2.18

$$\theta_0 = \min_{\substack{i \\ \text{such that} \\ x_{ij} > 0}} \left(\frac{x_{i0}}{x_{ij}}\right) = \frac{1}{3} \text{ for } i = l = 1$$

We now need to introduce a unit vector in Column $j = 1$, with the 1 in Row $l = 1$. We do this by dividing Row 1 by 3, adding to Row 3, and then multiplying by -2 and adding to Row 2. We usually represent this operation by circling the "pivot" element x_{lj} in the tableau, as in (2.24). The new tableau is

	x_1	x_2	x_3	x_4	x_5
$\frac{1}{3}$	1	$\frac{2}{3}$	$\frac{1}{3}$	0	0
$\frac{4}{3}$	0	$-\frac{7}{3}$	$-\frac{2}{3}$	1	0
$\frac{10}{3}$	0	$\frac{11}{3}$	$\frac{1}{3}$	0	1

The new basis is $\mathcal{B}' = \{A_1, A_4, A_5\}$, corresponding to the bfs $x_1' = \frac{1}{3}$, $x_4' = \frac{4}{3}$ and $x_5' = \frac{10}{3}$. In general, if x_{ij} and x_{ij}' are the old and new tableaux, respectively; $\mathcal{B}$ and $\mathcal{B}'$ the old and new basic sets, respectively; and the pivot element is x_{lj}, then

$$\begin{aligned} x_{lq}' &= \frac{x_{lq}}{x_{lj}} && q = 0, \ldots, n \\ x_{iq}' &= x_{iq} - x_{lq}' x_{ij} && i = 1, \ldots, m; \quad i \neq l \\ & && q = 0, \ldots, n \end{aligned} \tag{2.25}$$

$$B'(i) = \begin{cases} B(i) & i \neq l \\ j & i = l \end{cases}$$

Now that we know how to move from bfs to bfs, we need to investigate the effect of such moves on the cost.

2.6 Choosing a Profitable Column

The cost of a bfs x_0 with basis $\mathcal{B}$ is

$$z_0 = \sum_{i=1}^{m} x_{i0} c_{B(i)}$$

Now consider the process of bringing Column A_j into the basis: We write

A_j in terms of the basis columns as

$$A_j = \sum_{i=1}^{m} x_{ij} A_{B(i)} \tag{2.26}$$

This can be interpreted as meaning that for every unit of the variable x_j that enters the basis, an amount x_{ij} of each of the variables $x_{B(i)}$ must leave. Thus a unit increase in the variable x_j results in a net change in the cost equal to

$$c_j - \sum_{i=1}^{m} x_{ij} c_{B(i)}$$

The quantity on the right is important enough to be assigned its own symbol, z_j; and we call the difference

$$\bar{c}_j = c_j - z_j$$

the *relative cost* of Column j. It is then profitable to bring Column j into the basis exactly when $\bar{c}_j < 0$. Furthermore, when for all j, $\bar{c}_j \geq 0$, we are at a local optimum, which is also a global optimum. We prove all this in detail in the following.

First we introduce some vector and matrix notation. For any tableau X, let B be the $m \times m$ matrix comprised of the columns of A corresponding to the basis in X, and let c_B be the m-vector of costs corresponding to these basic variables. Then because the tableau X is obtained by diagonalizing the basic columns of A, we can write the tableau X as

$$X = B^{-1}A$$

and the vector $z = \text{col}\,(z_1, \ldots, z_n)$ from its definition as

$$z' = c'_B X = c'_B B^{-1} A$$

We use this matrix terminology again and again in what follows.

Theorem 2.8 (Optimality Criterion) *At a bfs x_0, a pivot step in which x_j enters the basis changes the cost by the amount*

$$\theta_0 \bar{c}_j = \theta_0 (c_j - z_j) \tag{2.27}$$

If

$$\bar{c} = c - z \geq 0 \tag{2.28}$$

then x_0 is optimal.

Proof From Eq. 2.19 in Theorem 2.7, the new solution is

$$x'_{i0} = \begin{cases} x_{i0} - \theta_0 x_{ij} & i \neq l \\ \theta_0 & i = l \end{cases}$$

so the new cost is

$$\begin{aligned} z'_0 &= \sum_{\substack{i=1 \\ i \neq l}}^{m} (x_{i0} - \theta_0 x_{ij}) c_{B(i)} + \theta_0 c_j \\ &= z_0 + \theta_0 (c_j - z_j) \end{aligned}$$

which establishes Eq. 2.27.

To show that $\bar{c} \geq 0$ implies that x_0 is optimal, let y be any feasible vector whatsoever, not necessarily basic. That is,

$$Ay = b$$

and

$$y \geq 0$$

Since $\bar{c} = c - z \geq 0$, the cost of y is

$$c'y \geq z'y = c'_B B^{-1}Ay = c'_B B^{-1}b = c'x_0$$

which shows that the cost of y can never be less than that of x_0. □

Since the values $\bar{c}_j$ tell us when a column can profitably enter the basis, we would like to keep them as part of the tableau. This is usually done in Row 0, as follows. Write the cost equation as

$$0 = -z + c_1x_1 + \cdots + c_nx_n \tag{2.29}$$

Now the relative cost $\bar{c}_j$ associated with a *basis* column j is

$$\bar{c}_j = c_j - z_j = c_j - \sum_{i=1}^{m} x_{ij}c_{B(i)} = 0$$

since x_{ij} is a unit vector with a 1 where $B(i) = j$. If we consider Eq. 2.29 the zeroth row of our tableau, we can make its components over basis columns zero by multiplying the ith row by $-c_{B(i)}$ and adding the result to the zeroth row. This yields in a nonbasic column the quantity

$$\bar{c}_j = c_j - \sum_{i=1}^{m} x_{ij}c_{B(i)}$$

and on the left-hand side of the zeroth equation

$$-z_0 = -\sum_{i=1}^{m} x_{i0}c_{B(i)}$$

The zeroth row therefore becomes

$$-z_0 = -z + \sum_{\substack{j=1 \\ A_j \notin \mathcal{B}}}^{n} \bar{c}_jx_j \tag{2.30}$$

If we now think of the tableau as a diagonal form in terms of the $m + 1$ variables $x_{B(1)}, x_{B(2)}, \ldots, x_{B(m)}$ and $-z$, we see that the same pivoting rules apply to the zeroth row as to Rows 1 to m. Hence we can carry along the relative costs $\bar{c}_j$ by keeping one more row in the tableau. There is, of course, no need to keep a column for the variable $-z$.

If we ignore for now the problem of degeneracy, then every pivot yields $\theta_0 > 0$, and we have a finite algorithm for LP, the *simplex algorithm*: If any $\bar{c}_j < 0$, pivot on Column j; when finally $\bar{c}_j \geq 0$ for all j, we have reached an optimal bfs. We never return to a previously visited bfs, because the cost decreases monotonically. Since there are a finite number of bfs's, we must terminate in a finite number of steps. We postpone the question of degeneracy to the next section, and conclude this section with an informal program (Fig. 2–3) and an example of the simplex algorithm.

```
procedure simplex
begin
  opt:= 'no', unbounded:= 'no';
  (comment: when either becomes 'yes' the algorithm terminates)
  while opt = 'no' and unbounded = 'no' do
    if c̄j ≥ 0 for all j then opt:= 'yes'
      else begin
          choose any j such that c̄j < 0;
          if xij ≤ 0 for all i then unbounded:= 'yes'
          else
            find θ0 = min      [xi0/xij] = xk0/xkj
                      i, xij>0
            and pivot on xkj
          end
end
```

Figure 2–3 The simplex algorithm.

Example 2.6

We consider the LP with the constraints of Sec. 2.5 and the cost function

$$z = x_1 + x_2 + x_3 + x_4 + x_5$$

The tableau of the original problem is therefore

	x_1	x_2	x_3	x_4	x_5
0	1	1	1	1	1
1	3	2	1	0	0
3	5	1	1	1	0
4	2	5	1	0	1

To start, we need a bfs, and we need to make zero the $\bar{c}_j$'s corresponding to the basic columns. We know from Sec. 2.5 that Columns 3, 4, and 5 yield a bfs. Subtracting Row 1 from Rows 2 and 3 and then subtracting the resulting Rows 1, 2, and 3 from Row 0 yields

		x_1	x_2	x_3 (basis)	x_4 (basis)	x_5 (basis)
$-z =$	-6	-3	-3	0	0	0
$x_3 =$	1	3	2	1	0	0
$x_4 =$	2	2	-1	0	1	0
$x_5 =$	3	-1	3	0	0	1

This represents the bfs indicated by the variables on the left, with cost $z = 6$. We have in Row 0, Columns 1 and 2, $\bar{c}_1 = -3$ and $\bar{c}_2 = -3$, respectively; so it is profitable for Column 1 or 2 to enter the basis. Choosing Column 2, we find

$$\theta_0 = \tfrac{1}{2} \quad \text{for} \quad l = 1$$

and we pivot on the element $x_{12} = 2$, which is circled. The resulting tableau is

		x_1	x_2	x_3	x_4	x_5
$-z =$	$-\frac{9}{2}$	$\frac{3}{2}$	0	$\frac{3}{2}$	0	0
$x_2 =$	$\frac{1}{2}$	$\frac{3}{2}$	1	$\frac{1}{2}$	0	0
$x_4 =$	$\frac{5}{2}$	$\frac{7}{2}$	0	$\frac{1}{2}$	1	0
$x_5 =$	$\frac{3}{2}$	$-\frac{11}{2}$	0	$-\frac{3}{2}$	0	1

which is optimal, with cost $z = \frac{9}{2}$. □

2.7 Pivot Selection and Bland's Anticycling Algorithm

There is a certain amount of uncertainty in the simplex algorithm as we described it: We have not said how to choose which column j (with $c_j - z_j < 0$) enters the basis; and we have not said how to resolve ties in the calculation of θ_0, which determines the row l and the variable $x_{B(l)}$ to leave the basis.

We first take up the question of column selection. Unfortunately, there is no theory to guide us here, and we must rely on empirical observations. The oldest and most widely used criterion is simply to choose the $\bar{c}_j < 0$ which is most negative. As we established above, a unit increase in the variable x_j entering the basis results in a change of $\bar{c}_j$ in the cost, so $\bar{c}_j$ can be thought of as the *derivative* of the cost with respect to distance in the *space of nonbasic variables*. Choosing the most negative $\bar{c}_j$ then corresponds to a kind of steepest descent policy, which is called the *nonbasic gradient* method [KQ]. By no means does this ensure, however, that the actual decrease in cost, $\theta_0 \bar{c}_j$, will be as large as possible, since we do not know θ_0 until we compute the ratios for row selection. This suggests another policy: Choose the column that results in the largest decrease in cost. This method, called the *greatest increment* method, carries with it an additional computational burden at each pivot step but offers the possibility of reaching optimality after a fewer number of pivots than the nonbasic gradient method.

A unit increase in the nonbasic variable x_j changes the entire vector x by

$$x_k = \begin{cases} +1 & k = j \\ -x_{ij} & k = B(i), \quad i = 1, \ldots, m \\ 0 & \text{otherwise} \end{cases}$$

We therefore can compute the derivative of the cost with respect to distance in the *space of all variables*,

$$\frac{\bar{c}_j}{\sqrt{1 + \sum_{i=1}^{m} x_{ij}^2}}$$

and the column selection policy corresponding to this derivative is called the *all-variable gradient* method.

Kuhn and Quandt [KQ] report the results of extensive computer experiments with these and other methods. The results, on problems with up to 25 rows, indicate that the all-variable gradient method converges in fewer pivots than the nonbasic gradient or greatest increment methods and is also faster. Goldfarb and Reid [GR] have described a fast way to compute the all-variable derivative and report good results with the all-variable gradient policy. The reader should view these results with some caution, however. First, the computation times reported by Kuhn and Quandt show at best no more than an improvement factor of two, and such improvements in running time can often result from changes in programming details. Second, the random class of LP's used for the tests may not reflect anybody's "typical" LP. Last, the nonbasic gradient method has the important advantage of simplicity and is still the most popular method actually programmed.

We now turn next to the resolution of ties in the row selection procedure. Here the possibility of degeneracy presents a certain danger: If we pivot during the simplex algorithm on element $x_{ij} > 0$ and the component x_{i0} of the bfs is zero, then $\theta_0 = 0$ and the cost increment $\theta_0(c_j - z_j) = 0$. That is, the cost z does not decrease, even though we choose a Column j with $c_j - z_j < 0$. It is further possible that we go through a sequence of such pivots, returning to our starting point. This means that the algorithm will loop indefinitely (assuming that the choices of column and row are made deterministically), and that would be a most undesirable situation. This phenomenon is called *cycling*.

Example 2.7 [Be1]

Consider the tableau

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
3	$-\frac{3}{4}$	$+20$	$-\frac{1}{2}$	$+6$	0	0	0
0	$\frac{1}{4}$	-8	-1	9	1	0	0
0	$\frac{1}{2}$	-12	$-\frac{1}{2}$	3	0	1	0
1	0	0	1	0	0	0	1

Let us pivot from this bfs with the following tie-breaking rules.

(a) Always select the nonbasic variable with the most negative $\bar{c}_j$ to enter the basis.

(b) In case of a tie, always select the basic variable with the smallest subscript to leave the basis.

We obtain the following sequence of tableaux (pivots are circled)

3	0	-4	$-\frac{7}{2}$	33	3	0	0
0	1	-32	-4	36	4	0	0
0	0	4	$\frac{3}{2}$	-15	-2	1	0
1	0	0	1	0	0	0	1

3	0	0	-2	18	1	1	0
0	1	0	8	-84	-12	8	0
0	0	1	$\frac{3}{8}$	$-\frac{15}{4}$	$-\frac{1}{2}$	$\frac{1}{4}$	0
1	0	0	1	0	0	0	1

3	$\frac{1}{4}$	0	0	-3	-2	3	0
0	$\frac{1}{8}$	0	1	$-\frac{21}{2}$	$-\frac{3}{2}$	1	0
0	$-\frac{3}{64}$	1	0	$\frac{3}{16}$	$\frac{1}{16}$	$-\frac{1}{8}$	0
1	$-\frac{1}{8}$	0	0	$\frac{21}{2}$	$\frac{3}{2}$	-1	1

3	$-\frac{1}{2}$	16	0	0	-1	1	0
0	$-\frac{5}{2}$	56	1	0	2	-6	0
0	$-\frac{1}{4}$	$\frac{16}{3}$	0	1	$\frac{1}{3}$	$-\frac{2}{3}$	0
1	$\frac{5}{2}$	-56	0	0	-2	6	1

3	$-\frac{7}{4}$	44	$\frac{1}{2}$	0	0	-2	0
0	$-\frac{5}{4}$	28	$\frac{1}{2}$	0	1	-3	0
0	$-\frac{1}{6}$	-4	$-\frac{1}{6}$	1	0	$\frac{1}{3}$	0
1	0	0	1	0	0	0	1

3	$-\frac{3}{4}$	$+20$	$-\frac{1}{2}$	6	0	0	0
0	$\frac{1}{4}$	-8	-1	9	1	0	0
0	$\frac{1}{2}$	-12	$-\frac{1}{2}$	3	0	1	0
1	0	0	1	0	0	0	1

Then, after six pivots, *we arrive at the same bfs with which we started.* All intermediate pivots introduced new basic variables at zero level, and there was no change in the cost. We say that simplex (with this particular pivot rule) has *cycled.* □

We can view our problem at this point as one of resolving the uncertainties in the simplex algorithm in such a way as to prevent cycling. It is sometimes reported that cycling simply does not occur in practice, even though artificial examples can be constructed. This is contradicted by a recent report [KS]. In addition, some LP formulations of combinatorial problems are highly degenerate, and it is not at all clear that we can trust to luck to avoid cycling, aesthetic considerations aside.

The simplest way to avoid cycling is to resolve ties in a random way—with probability 1 we shall escape from any loop. This complicates the programming of the ratio test, however, and is not as intellectually satisfying as a deterministic rule that guarantees finiteness of the simplex algorithm. It does not seem to be a popular policy.

In one standard approach to cycle avoidance, any choice of column is allowed, and ties in the θ_0 calculation are resolved in such a way as to ensure that the zeroth row increases lexicographically, thus ensuring that no basis is ever repeated. This has the advantage of allowing any column selection policy. We postpone a description of this method until Chapter 14. We shall describe here a relatively recent algorithm that prevents cycling, due to R.G. Bland [Bl], which is remarkable in its simplicity. We first need the following lemma.

Lemma 2.3 *Let $\bar{c}'$ be the relative cost row for any tableau X_1 with a unit basis, not necessarily corresponding to a feasible solution. (That is, some x_{i0} in the zeroth column can be negative.) Let y be any solution to the constraints $Ay = b$, not necessarily corresponding to a feasible solution. (That is, some y_j can be negative.) Let f be the cost associated with X_1 and g with y. Then*

$$\bar{c}'y = g - f$$

Proof A direct calculation yields

$$\bar{c}'y = (c' - z')y = c'y - z'y = g - c'_B B^{-1}Ay = g - c'_B B^{-1}b = g - f$$

since $B^{-1}b$ is the zeroth column of tableau X_1. □

Theorem 2.9 (Bland's anticycling algorithm [Bl]) *Suppose in the simplex algorithm we choose the column to enter the basis by*

$$j = min\{j: c_j - z_j < 0\}$$

(choose the lowest numbered favorable column), and the row by

$$B(i) = min\left\{B(i): x_{ij} > 0 \quad and \quad \frac{x_{i0}}{x_{ij}} \le \frac{x_{k0}}{x_{kj}} \quad for\ every\ k\ with\ x_{kj} > 0\right\}$$

(choose in case of tie the lowest numbered column to leave the basis). Then the algorithm terminates after a finite number of pivots.

***Proof*†** We find a contradiction from the assumption that a cycle exists. For a cycle to occur, there must be a finite sequence of pivots that returns to a bfs. The cost z remains constant during this cycle, and the x_{i0} associated with each pivot must be zero, for otherwise $\theta_0 > 0$, which would imply that z decreases. This in turn implies that the zeroth column x_{i0}, $i = 1, \ldots, m$ remains constant during the cycle.

Throw away those rows and columns not containing pivots during the cycle, yielding a new program which still cycles and has all $x_{i0} = 0$ and constant z during the cycle.

Now let q be the *largest* index of a variable entering the basis during the cycle, and consider two tableaux: T_1, the tableau when x_q is about to enter the basis; and T_2, the tableau when x_q is about to leave the basis (see Fig. 2–4). Denote the entries in T_1 by x_{ij}, with basis $\mathcal{B}$, and in T_2 by $\hat{x}_{ij}$, with basis $\hat{\mathcal{B}}$, and let column p be the column entering in T_2. We now apply Lemma 2.3 by

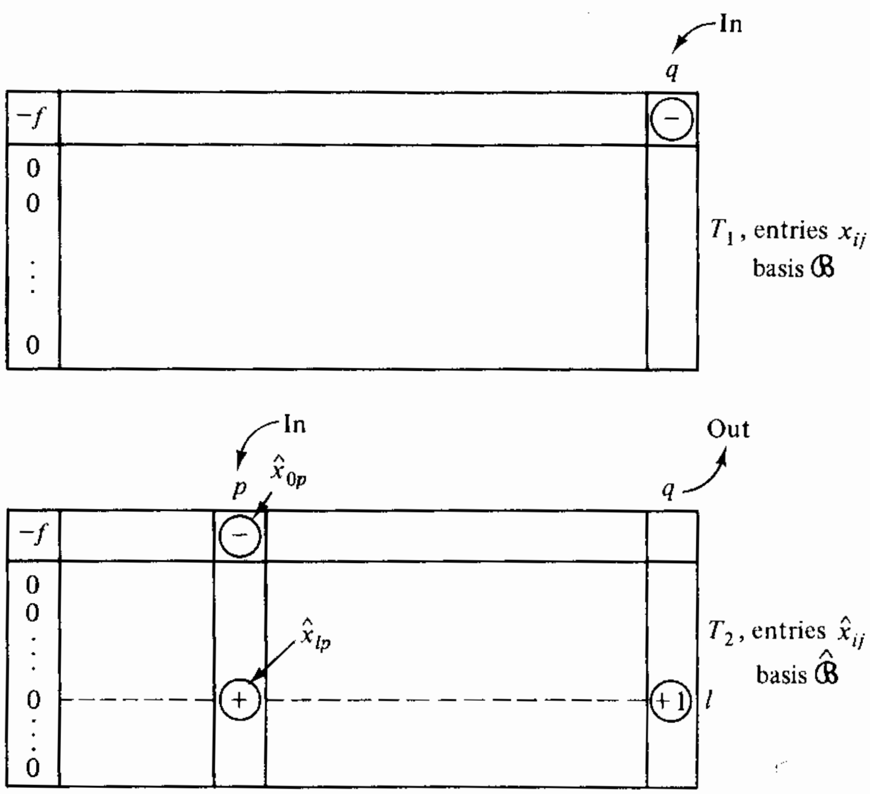


Figure 2–4 The tableaux T_1 and T_2 in the proof of Theorem 2.9. The variable x_q is about to enter in T_1 and leave in T_2.

†This proof is a simplification of Bland's proof due to H. W. Kuhn [Ku].

constructing two solutions: In T_1 we use simply x_0, the bfs, and identify T_1 with X_1. From T_2, we define a solution y by

$$y_j = \begin{cases} 1 & \text{if } j = p \\ -\hat{x}_{ip} & \text{if } A_j \in \hat{\mathcal{B}} \\ 0 & \text{otherwise} \end{cases}$$

Notice that y is neither basic nor feasible but is a solution of $Ay = b$, and hence it satisfies the requirements of Lemma 2.3. Furthermore, the cost of y is $f + \hat{x}_{0p}$, so the conclusion of Lemma 2.3 gives

$$\bar{c}'y = \hat{x}_{0p} < 0$$

The inequality follows since Column p is entering in T_2 and therefore must have negative relative cost $\hat{x}_{0p}$.

Now by the choice of pivot column in T_1,

$$\bar{c}_j \begin{cases} \geq 0, & j < q \\ < 0, & j = q \end{cases}$$

and by the choice of pivot row in T_2

$$y_j = \begin{cases} -\hat{x}_{ip} < 0, & j = q \\ 0,\ 1, \text{ or } -\hat{x}_{ip} \geq 0, & j < q \end{cases}$$

Therefore

$$\bar{c}'y = \sum_{j<q} \bar{c}_j y_j + \bar{c}_q y_q \geq \bar{c}_q y_q > 0$$

which is a contradiction. □

2.8 Beginning the Simplex Algorithm

We are left with only one detail: How do we obtain an initial bfs with which to start the simplex algorithm? Sometimes, of course, we may inherit a bfs as part of the problem formulation. For example, we may begin with inequalities of the form $Ax \leq b$, in which case the slack variables constitute a bfs. If we are not so lucky, we may use the *artificial variable*, or *two-phase*, method. In this method we simply append new, "artificial" variables x_i^a, $i = 1, \ldots, m$ to the left of the tableau as follows.

$$\begin{array}{c|ccc|ccc|}
 & x_1^a & \cdots & x_m^a & x_1 & \cdots & x_n \\
\hline
 & 1 & & 0 & & & \\
b & & \ddots & & & A & \\
 & 0 & & 1 & & & \\
\hline
\end{array}
\qquad
\begin{array}{ll}
x_j \geqslant 0 & j = 1, \ldots, n \\
x_i^a \geqslant 0 & i = 1, \ldots, m
\end{array}$$

We multiplied some of the original equations by -1 when necessary to make $b \geq 0$. We then have a bfs $x_i^a = b_i$.

In *Phase I*, we minimize the cost function

$$\xi = \sum_{i=1}^{m} x_i^a$$

subject to the above constraints, using the simplex algorithm. There are three possible outcomes.

Case 1 We reduce ξ to zero, and all the x_i^a are driven out of the basis; in this case we now have a bfs to the original problem.

Case 2 We reach optimality with $\xi > 0$, in which case we know that the original problem violates Assumption 2.2—that there is some feasible solution. (If there were a feasible solution to the original problem, it would show that the minimum value of ξ is zero.)

Case 3 We reduce ξ to zero, but some artificial variables remain in the basis at *zero level.*

In Case 1 the columns corresponding to the artificial variables can be dropped and we can continue directly with *Phase II*: The ordinary simplex algorithm using the original cost function $z = c'x$. It is sometimes convenient to start with two cost rows, one for ξ and one for z. When we switch from Phase I to II, we simply change from the first cost row to the second. In Case 2, of course, we must simply stop.

In Case 3 suppose that the ith column of the basis at the end of Phase I is the column corresponding to an artificial variable, and $x_{i0} = 0$. We may pivot on any nonzero (not necessarily positive) element x_{ij} of Row i corresponding to a non-artificial variable. Since θ_0 will be zero, no infeasibility or change in cost ξ will result. This is not exactly pivoting, since it might be the case that $x_{ij} < 0$ or $\bar{c}_j > 0$; we simply say that we are *driving the artificial variable out of the basis.* We repeat this until we obtain a feasible basis with the original variables. The only way that this can fail is that a row can be zero in all the columns corresponding to non-artificial variables. But this means that we have arrived at a zero row in the original matrix by elementary row operations, which contradicts Assumption 2.1 and shows that A was not of full rank m. We can delete such zero rows and continue in Phase II with a basis of lower dimension.

Conversely, if the original set of equations is not of full rank m, we cannot reach Case 1. We shall therefore reach Case 2 if the problem has no feasible solution, or an artificial variable will remain in the basis at zero level at the end of Phase I and, in fact, at the end of Phase II.

Example 2.8

In Example 2.6 we knew a set of basic columns a priori. To use the two-phase method, we would begin with the tableau

		x_1^a	x_2^a	x_3^a	x_1	x_2	x_3	x_4	x_5	
$-z =$	0	0	0	0	1	1	1	1	1	row 0′
$-\xi =$	0	1	1	1	0	0	0	0	0	row 0
	1	1	0	0	3	2	1	0	0	
	3	0	1	0	5	1	1	1	0	
	4	0	0	1	2	5	1	0	1	

We subtract Rows 1, 2, and 3 from the ξ cost row, Row 0, to begin with zero relative costs for our original basis x_1^a, x_2^a, and x_3^a; this yields

		x_1^a	x_2^a	x_3^a	x_1	x_2	x_3	x_4	x_5
$-z =$	0	0	0	0	1	1	1	1	1
$-\xi =$	-8	0	0	0	-10	-8	-3	-1	-1
$x_1^a =$	1	1	0	0	3	2	1	0	0
$x_2^a =$	3	0	1	0	5	1	1	1	0
$x_3^a =$	4	0	0	1	2	5	1	0	1

The successive pivots and tableaux in Phase I are shown below.

		x_1^a	x_2^a	x_3^a	x_1	x_2	x_3	x_4	x_5
$-z =$	$-\frac{1}{3}$	$-\frac{1}{3}$	0	0	0	$\frac{1}{3}$	$\frac{2}{3}$	1	1
$-\xi =$	$-\frac{14}{3}$	$\frac{10}{3}$	0	0	0	$-\frac{4}{3}$	$\frac{1}{3}$	-1	-1
$x_1 =$	$\frac{1}{3}$	$\frac{1}{3}$	0	0	1	$\frac{2}{3}$	$\frac{1}{3}$	0	0
$x_2^a =$	$\frac{4}{3}$	$-\frac{5}{3}$	1	0	0	$-\frac{7}{3}$	$-\frac{2}{3}$	1	0
$x_3^a =$	$\frac{10}{3}$	$-\frac{2}{3}$	0	1	0	$\frac{11}{3}$	$\frac{1}{3}$	0	1

		x_1^a	x_2^a	x_3^a	x_1	x_2	x_3	x_4	x_5
$-z =$	$-\frac{1}{2}$	$-\frac{1}{2}$	0	0	$-\frac{1}{2}$	0	$\frac{1}{2}$	1	1
$-\xi =$	-4	4	0	0	2	0	1	-1	-1
$x_2 =$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$\frac{3}{2}$	1	$\frac{1}{2}$	0	0
$x_2^a =$	$\frac{5}{2}$	$-\frac{1}{2}$	1	0	$\frac{7}{2}$	0	$\frac{1}{2}$	1	0
$x_3^a =$	$\frac{3}{2}$	$-\frac{15}{6}$	0	1	$-\frac{11}{2}$	0	$-\frac{3}{2}$	0	1

		x_1^a	x_2^a	x_3^a	x_1	x_2	x_3	x_4	x_5
$-z =$	-3	0	-1	0	-4	0	0	0	1
$-\xi =$	$-\frac{3}{2}$	$\frac{7}{2}$	1	0	$\frac{11}{2}$	0	$\frac{3}{2}$	0	-1
$x_2 =$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$\frac{3}{2}$	1	$\frac{1}{2}$	0	0
$x_4 =$	$\frac{5}{2}$	$-\frac{1}{2}$	1	0	$\frac{7}{2}$	0	$\frac{1}{2}$	1	0
$x_3^a =$	$\frac{3}{2}$	$-\frac{5}{2}$	0	1	$-\frac{11}{2}$	0	$-\frac{3}{2}$	0	1

$-z =$	$-\frac{9}{2}$	$\frac{5}{2}$	-1	-1	$\frac{3}{2}$	0	$\frac{3}{2}$	0	0
$-\xi =$	0	1	1	1	0	0	0	0	0
$x_2 =$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$\frac{3}{2}$	1	$\frac{1}{2}$	0	0
$x_4 =$	$\frac{5}{2}$	$-\frac{1}{2}$	1	0	$\frac{7}{2}$	0	$\frac{1}{2}$	1	0
$x_5 =$	$\frac{3}{2}$	$-\frac{5}{2}$	0	1	$-\frac{11}{2}$	0	$-\frac{3}{2}$	0	1

At the end of Phase I, $\xi = 0$, and the resulting tableau is in fact optimal for Phase II as well. (The final tableau for variables x_1 to x_5 agrees with the final, optimal tableau in Example 2.6.) □

The final two-phase algorithm is shown in Fig. 2–5. Notice that we have obviated Assumptions 2.1–2.3: (1) If the matrix A of the original problem is

```
               procedure two-phase
               begin
                 infeasible:= 'no', redundant:= 'no';
                 (comment: Phase I may set these to 'yes')
Phase I:         introduce an artificial basis, x_i^a;
                 call simplex with cost ξ = Σ x_i^a;
                 if ξ_opt > 0 in Phase I then infeasible:= 'yes'
                   else begin
                       if an artificial variable is in the basis and
                         cannot be driven out then redundant:= 'yes',
                         and omit the corresponding row;
Phase II:              call simplex with original cost
                       end
               end
```

Figure 2–5 The final two-phase algorithm.

not of rank m, we learn so at the end of Phase I and can continue; (2) if the original problem is infeasible, we also learn that at the end of Phase I; and (3) if the problem is of unbounded cost, we learn about it in Phase II.

2.9
Geometric Aspects of Pivoting

Let us solve the LP of Example 2.2 by simplex and trace the sequence of bfs's obtained on the corresponding polytope. The resulting sequence of tableaux is shown in Fig. 2–6, together with the sequence of vertices of the polytope corresponding to the bfs's produced. We observe that simplex simply traces a path along edges of the polytope. We shall next prove formally that this is always the case.

−34	−1	−14	−6	0	0	0	0
4	1	1	1	1	0	0	0
2	1	0	0	0	1	0	0
3	0	0	1	0	0	1	0
6	0	3	1	0	0	0	1

1

−32	0	−14	−6	0	1	0	0
2	0	1	1	1	−1	0	0
2	1	0	0	0	1	0	0
3	0	0	1	0	0	1	0
6	0	3	1	0	0	0	1

2

−20	0	−8	0	6	−5	0	0
2	0	1	1	1	−1	0	0
2	1	0	0	0	1	0	0
1	0	−1	0	−1	1	1	0
4	0	2	0	−1	1	0	1

3

−4	0	0	8	14	−13	0	0
2	0	1	1	1	−1	0	0
2	1	0	0	0	1	0	0
3	0	0	1	0	0	1	0
0	0	0	−2	−3	3	0	1

4

-4	0	0	$-\frac{2}{3}$	1	0	0	$\frac{13}{3}$	
2	0	1	$\frac{1}{3}$	0	0	0	$\frac{1}{3}$	
2	1	0	$\frac{2}{3}$	1	0	0	$-\frac{1}{3}$	5
3	0	0	1	0	0	1	0	
0	0	0	$-\frac{2}{3}$	-1	1	0	$\frac{1}{3}$	

-2	1	0	0	2	0	0	4	
1	$-\frac{1}{2}$	1	0	$-\frac{1}{2}$	0	0	$\frac{1}{2}$	
3	$\frac{3}{2}$	0	1	$\frac{3}{2}$	0	0	$-\frac{1}{2}$	6
0	$-\frac{3}{2}$	0	0	$-\frac{3}{2}$	0	1	$\frac{1}{2}$	
2	1	0	0	0	1	0	0	

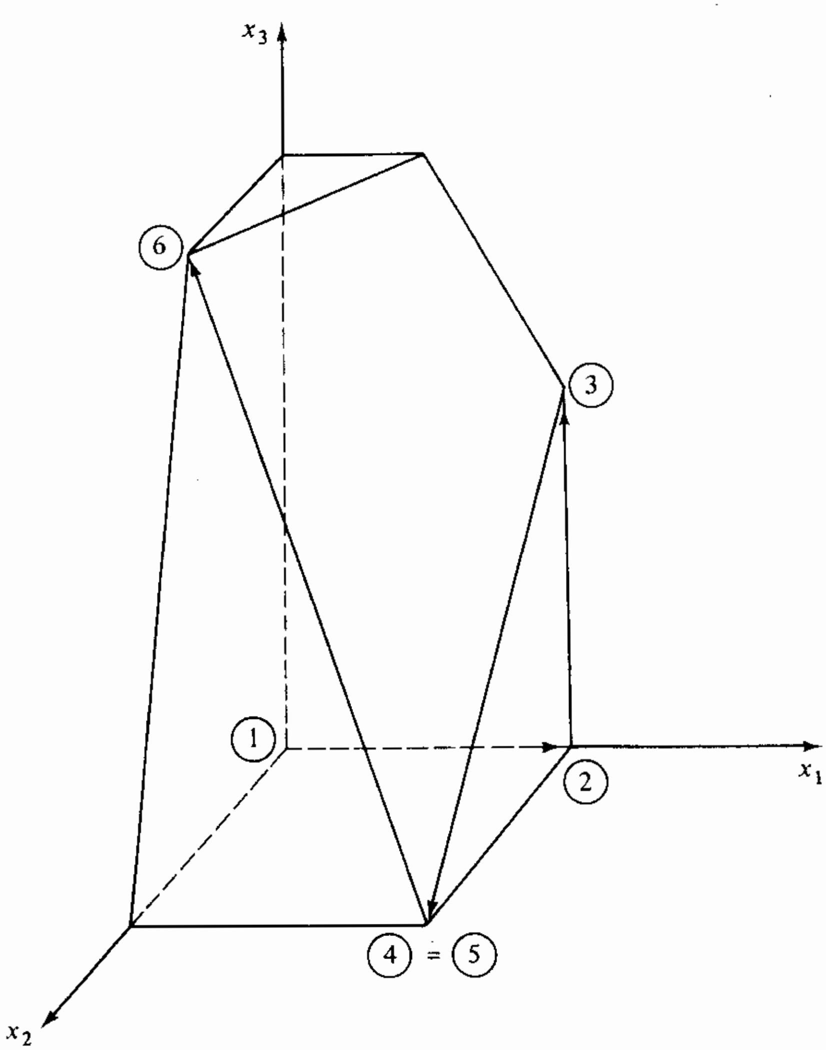


Figure 2–6

Definition 2.6

Two vertices $\hat{x}$ and $\hat{y}$ of a polytope are called *adjacent* if the line segment $[\hat{x}, \hat{y}]$ is an edge of the polytope. Two distinct bfs's x and y of an LP $Ax = b$, $x \geq 0$ are called *adjacent* if there exist bases $\mathcal{B}_x$, $\mathcal{B}_y$ such that $\mathcal{B}_y = (\mathcal{B}_x - \{A_j\}) \cup \{A_k\}$ and $x = B_x^{-1}b$, $y = B_y^{-1}b$. $\square$

Thus simplex proceeds by replacing one bfs with another adjacent one, having no greater cost, until the optimal bfs is obtained.

We can now prove the following extension of Theorem 2.4 to edges.

Theorem 2.10 *Let P be a polytope, $F = \{x\colon Ax = b, x \geq 0\}$ the corresponding feasible set, and $\hat{x} = (x_1, \ldots, x_{n-m})$, $\hat{y} = \{y_1, \ldots, y_{n-m}\}$ be distinct vertices of P. Then the following are equivalent.*

(a) *The segment $[\hat{x}, \hat{y}]$ is an edge of P.*

(b) *For every $\hat{z} \in [\hat{x}, \hat{y}]$, if $\hat{z} = \lambda\hat{z}' + (1 - \lambda)\hat{z}''$ with $0 < \lambda < 1$ and $\hat{z}'$, $\hat{z}'' \in P$, then $\hat{z}', \hat{z}'' \in [\hat{x}, \hat{y}]$.*

(c) *The corresponding vectors x, y of F are adjacent bfs's.*

Proof (a) $\Rightarrow$ (b) If $[\hat{x}, \hat{y}]$ is an edge of P, then there is a supporting hyperplane H with equation, say, $h'\hat{x} = g$. Every $\hat{z} \in [\hat{x}, \hat{y}]$ therefore satisfies $h'\hat{z} = g$. Now, assume that $\hat{z} = \lambda\hat{z}' + (1 - \lambda)\hat{z}''$ with $1 < \lambda < 0$, $\hat{z}', \hat{z}'' \in P$ but not both in $[\hat{x}, \hat{y}]$. Thus $h'\hat{z}' \leq g$, $h'\hat{z}'' \leq g$, and one inequality is strict. Therefore, $h'\hat{z} = h'(\lambda\hat{z}' + (1 - \lambda)\hat{z}'') < g$, a contradiction.

(b) $\Rightarrow$ (c) Assume that bfs's $x, y \in F$ correspond to points in P with Property (b), but are nonadjacent.

Let $\mathcal{M}_x$ and $\mathcal{M}_y$ be the sets of columns corresponding to nonzero components of x and y, respectively. Now it is easy to see that there is a bfs $w \neq x, y$ with nonzero components only in $\mathcal{M}_x \cup \mathcal{M}_y$. Otherwise, we could have a cost vector

$$c_j = \begin{cases} 0 & A_j \in \mathcal{M}_y \\ 1 & A_j \in \mathcal{M}_x - \mathcal{M}_y \\ nM & \text{otherwise} \end{cases}$$

where M is a suitably large number, say the one defined in Lemma 2.1. Then y is uniquely optimal, and any feasible solution with nonzero components out of $\mathcal{M}_x \cup \mathcal{M}_y$ has cost more than x. Thus simplex started at x would fail to discover a sequence of adjacent bfs's with nonincreasing cost leading to the optimum, which is absurd. So such a $w \neq x, y$ does exist, and, furthermore, $\hat{w}$ does not lie on $[\hat{x}, \hat{y}]$ because the points $\hat{w}, \hat{x}, \hat{y}$ correspond to distinct vertices of the polytope P.

Now let $z = \frac{1}{2}(x + y)$ and consider the difference

$$d = z - w$$

It is nonzero only for columns in $\mathcal{M}_x \cup \mathcal{M}_y$, and hence there exists a positive number θ such that

$$u_1 = z + \theta d$$

and

$$u_2 = z - \theta d$$

are feasible. Hence $z = \frac{1}{2}(u_1 + u_2)$, where $\hat{u}_1$ and $\hat{u}_2$ do not lie on $[\hat{x}, \hat{y}]$; this contradicts Property (b).

(c) $\Rightarrow$ (a) Let $\mathcal{B}_x, \mathcal{B}_y$ be the bases corresponding to x and y, respectively, with $\mathcal{B}_y = \mathcal{B}_x \cup \{A_j\} - \{A_k\}$ for some columns A_j, A_k. Let us construct a cost vector c by

$$c_j = \begin{cases} 0 & \text{if } A_j \in \mathcal{B}_y \cup \mathcal{B}_x \\ 1 & \text{otherwise} \end{cases}$$

All feasible solutions that are convex combinations of x and y are optimal. Furthermore, these are the only optimal solutions. To show this suppose that z is optimal. Then z is, by Theorem 2.3, a convex combination of bfs's, and, in particular, of bfs's with bases subsets of $\mathcal{B}_x \cup \mathcal{B}_y$; however, x and y are the only such bfs's.

It follows that only convex combinations w of x and y satisfy $Aw = b$, $w \geq 0$ and $c'w \leq c'x$. Therefore, in P, only points $\hat{w}$ on the segment $[\hat{x}, \hat{y}]$ satisfy

$$d'\hat{w} \leq d'\hat{x}$$

where d is defined, as in the proof of Theorem 2.4, to be

$$d_i = c_i - \sum_{j=1}^{m} h_{n-m+j,i} c_{n-m+j}$$

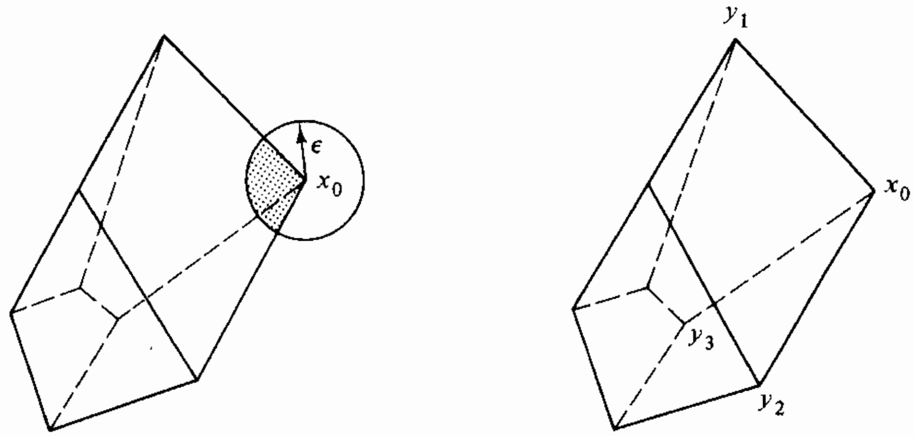
Hence $[\hat{x}, \hat{y}]$ is the intersection of a halfspace with P and is therefore an edge. $\square$

A final comment on simplex: By our discussion of Chapter 1, LP is a convex programming problem, and so the Euclidean neighborhood N_ϵ is exact. That is, if we search in the neighborhood of all points in F that are within ϵ of some $x_0 \in F$ and find no solution better than x_0, then x_0 is globally optimal (see Figure 2–7(a)).

The simplex algorithm has revealed another exact neighborhood, combinatorially and computationally much more meaningful. First, we do not have to consider all of the (uncountably infinite) set F, but just the finite set of basic feasible solutions. Furthermore, within this set of bfs's, we have the neighborhood

$$N_A(x_0) = \{y : y \text{ is a bfs adjacent to } x_0\}$$

Then Theorem 2.8 tells us that N_A is *exact* for LP (see Figure 2–7(b)). What is more, $N_A(x_0)$ contains a few (at most $n - m$) bfs's and can be searched very fast—in fact, just by looking at the signs of the $\bar{c}_j$'s. Thus simplex can be viewed



(a) (b)

Figure 2–7 (a) The exact neighborhood N_ϵ. (b) The exact neighborhood $N_A(x_0) = \{y_1, y_2, y_3\}$.

as just a clever implementation of the general neighborhood search scheme for the exact neighborhood structure N_A.

PROBLEMS

1. Show that the converse of Theorem 2.5 is not true; that is, that there can exist a degenerate vertex whose corresponding basis is unique.
2. Show that a polytope F defined by an instance of LP is a closed set.
3. Suppose in an instance of LP, we have n variables that are unconstrained in sign. Show how they can be replaced by $n + 1$ variables that are constrained to be nonnegative.
4. Check the statement in the proof of Theorem 2.4 that the set $\mathcal{B}$ can be augmented to a basis, and the similar statement in the proof of Theorem 2.1.

*5. Show that the optimality criterion of Theorem 2.8 is not necessary at an optimal vertex.

*6. Show that the condition $\theta_0 = 0$ for every possible pivot in the simplex algorithm does not imply optimality.

*7. Show that a linear program cannot cycle unless we have at least two basic variables that are zero.

8. Show that the set of optimal points of an instance of LP is a convex set.
9. We are given the following instance A of LP in standard form:

$$\min c'x$$
$$Ax = b$$
$$x \geq 0$$

We also have instance B:

$$\begin{aligned} \min\ & -c'x \\ Ax &= b \\ x &\geq 0 \end{aligned}$$

Can instances A and B both have feasible solutions with arbitrarily small cost? If yes, give an example; if not, prove so.

10. Show that the set G in the proof of Theorem 2.2 is closed. (*Hint:* Show that any point outside G has a neighborhood outside G.)

11. Does the fact that every vertex of an LP is nondegenerate imply that the solution is unique? If so, prove it; if not, give a counterexample.

12. Answer yes or no and prove your answer: Can a pivot of the simplex algorithm move the feasible point a positive distance in R^n while leaving the cost unchanged?

13. Can a vector which has just left the basis in the simplex algorithm reenter on the very next pivot?

14. The following fragment of FORTRAN code calculates $\bar{c}_j$ for pricing in the simplex algorithm and decides whether to pivot in order to bring Column j into the basis:

```
      .
      .
      .
      CBAR = C(J)
      DO 1  I = 1,M
    1 CBAR = CBAR - C(BASIS(I))*X(I,J)
      IF(CBAR.LT.0.)GO TO 3
      .
      .
      .
```

Assume the variables C, *BASIS*, and X are defined appropriately at this point in the program. Give a reason why this will not work well in practice, and suggest a simple alteration which will. (The issue here is not language dependent.)

15. (Programming project) Write a computer program that implements the two-phase simplex algorithm for an LP in standard form. The input should be the vectors b, and c, and the matrix A. The program should terminate in one of the following four ways.

1. Unbounded solution found in Phase I. This is impossible (why?), but should be a logical branch in the program as an error check.
2. Optimal solution found in Phase I with positive cost. This means the original problem is infeasible.
3. Unbounded solution found in Phase II. This means that the original problem has unbounded cost.
4. Optimal solution found in Phase II. This means that the original problem has been solved.

Your program should print out:

(a) The problem data;

(b) The row, column, and cost after each pivot in both phases;

(c) A message after Phase I and after Phase II if entered;

(d) The final basis, tableau, and cost, regardless of the termination point.

Test your program on problems that terminate in as many ways as possible.

16. Prove the following: If F is a k-dimensional face of a convex polytope P in R^d, then F is also a convex polytope, and furthermore every vertex of F is also a vertex of P.

17. Prove: If an LP is unbounded, then there is a rational vector α such that (*a*) $c'\alpha < 0$, and (*b*) if x is feasible and $k > 0$, then $x + k\alpha$ is also feasible.

NOTES AND REFERENCES

The simplex algorithm was invented in 1947 by G. B. Dantzig, and we cannot recommend too highly his comprehensive text

[Da1] DANTZIG, G. B., *Linear Programming and Extensions*. Princeton, N.J.: Princeton University Press, 1963.

The reader will find there a detailed and first-hand account of the origins of linear programming, as well as a development of the simplex algorithm and its variations. Besides this, there are many other excellent texts devoted to linear programming. Among them are

[BJ] BAZARAA, M. S., and J. J. JARVIS, *Linear Programming and Network Flows*. New York: John Wiley & Sons, Inc., 1977.

[CS] COOPER, L., and D. STEINBERG, *Methods and Applications of Linear Programming*. Philadelphia: W.B. Saunders, 1974.

[Gal] GALE, D., *The Theory of Linear Economic Models*. New York: McGraw-Hill Book Company, 1960.

[Gas] GASS, S. I., *Linear Programming* (4th ed.). New York: McGraw-Hill Book Company, 1975.

[Had2] HADLEY, G., *Linear Programming*. Reading, Mass.: Addison-Wesley Publishing Co., Inc., 1962.

[Hu] HU, T. C., *Integer Programming and Network Flows*. Reading, Mass.: Addison-Wesley Publishing Co., Inc., 1970.

[Si] SIMONNARD, M., *Linear Programming* (translated from the French by W. S. Jewell). Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1966.

[YG] YUDIN, D. B., and E. G. GOL'SHTEIN, *Linear Programming* (translated from the Russian by Z. Lerman). Jerusalem: Israel Program for Scientific Translations, 1965.

Dantzig [Da1] attributes the formulation of the diet problem to

[Sti] STIGLER, G. J., "The Cost of Subsistence," *J. Farm Econ.*, 27, no. 2 (May 1945), 303–14.

He also gives the first publication of the simplex algorithm as

[Da2] DANTZIG, G. B., "Programming of Interdependent Activities, II, Mathematical Model," pp. 19–32, in *Activity Analysis of Production and Allocation*, ed. T. C. Koopmans. New York: John Wiley & Sons, Inc., 1951. Also in *Econometrics* 17, nos. 3 and 4 (July–Oct. 1949), 200–11.

Theorem 2.3 can be considered a special case of the general fact that any closed, bounded convex set is the convex hull of its extreme points. For much more about polytopes, see

[Gru] GRÜNBAUM, B., *Convex Polytopes*. New York: John Wiley & Sons, Inc., 1967.

[Roc] ROCKAFELLAR, R. T., *Convex Analysis*. Princeton, N.J.: Princeton University Press, 1970.

Cycling in practical problems is described in

[KS] KOTIAH, T. C. T., and D. I. STEINBERG, "On the Possibility of Cycling with the Simplex Method," *OR*, 26, no. 2 (March–April 1978), 374–6.

The anticycling rule given in Section 2.7 is due to

[Bl] BLAND, R. G., "New Finite Pivoting Rules," Discussion Paper 7612, Center for Operations Research and Econometrics (CORE), Université Catholique de Louvain, Heverlee, Belgium, June 1976 (revised January 1977).

The proof given is after

[Ku] KUHN, H. W., Class Notes, Princeton University, 1976.

Computational experiments comparing different column selection rules are described in

[KQ] KUHN, H. W., and R. E. QUANDT, "An Experimental Study of the Simplex Method," pp. 107–24, in *Proceedings of Symposia on Applied Mathematics*, vol. XV, ed. N. Metropolis and others. American Mathematical Society, Providence, R.I.; 1963.

An all-variable steepest-descent method is described in

[GR] GOLDFARB, D., and J. K. REID, "A Practicable Steepest-Edge Simplex Algorithm, *Math. Prog.*, 12, no. 3 (June 1977), 361–71.

The cycling example is from

[Be1] BEALE, E. M. L., "Cycling in the Dual Simplex Algorithm," *Naval Research Logistics Quarterly*, 2, no. 4 (1955), 269–75.

18

Branch-and-Bound and Dynamic Programming

18.1 Branch-and-Bound for Integer Linear Programming

The branch-and-bound method is based on the idea of intelligently enumerating all the feasible points of a combinatorial optimization problem. The qualification *intelligently* is important here because, as should be clear by now, it is hopeless simply to look at all feasible solutions. Perhaps a more sophisticated way of describing the approach is to say that we try to construct a proof that a solution is optimal, based on successive partitioning of the solution space. The *branch* in branch-and-bound refers to this partitioning process; the *bound* refers to lower bounds that are used to construct a proof of optimality without exhaustive search. We shall develop the method in this section for ILP, and then put things in a more abstract framework.

Consider, then, the ILP

$$\begin{aligned} &\min z = c'x = c(x) \\ \textit{Problem 0} \qquad &Ax \leq b \qquad\qquad (18.1) \\ &x \geq 0, \text{ integer} \end{aligned}$$

If we solve the LP relaxation, we obtain a solution x^0, which in general is not integer. The cost $c(x^0)$ of this solution is, however, a lower bound on the optimal cost $c(x^*)$ (where x^* is the optimal solution to Problem 0), and if x^0 were integer, we would in fact be done. In the cutting-plane algorithm, we would now add a constraint to the relaxed problem that does not exclude feasible solutions to (18.1). Here, however, we are going to split the problem into two subproblems by adding two *mutually exclusive* and *exhaustive* constraints. Suppose that component x_i^0 of x^0 is noninteger, for example. Then the two subproblems are

$$\textit{Problem 1} \qquad \begin{aligned} \min z = c'x &= c(x) \\ Ax &\le b \\ x &\ge 0, \text{ integer} \\ x_i &\le \lfloor x_i^0 \rfloor \end{aligned} \tag{18.2}$$

and

$$\textit{Problem 2} \qquad \begin{aligned} \min z = c'x &= c(x) \\ Ax &\le b \\ x &\ge 0, \text{ integer} \\ x_i &\ge \lfloor x_i^0 \rfloor + 1 \end{aligned} \tag{18.3}$$

Example 18.1

A simple ILP is shown in Fig. 18–1(a); the solution is $x^* = (2, 1)$ and $c(x^*) = -(x_1 + x_2) = -3$. The initial relaxed problem has the solution $x^0 = (\frac{3}{2}, \frac{5}{2})$ with cost $c(x^0) = -4$. Figure 18–1(b) shows the two subproblems generated by choosing the noninteger component $x_1^0 = \frac{3}{2}$ and introducing the constraints

$$x_1 \le 1 \quad \text{and} \quad x_1 \ge 2 \quad \square$$

The solution to the original problem must lie in the feasible region of one of these two problems, simply because one of

$$\begin{aligned} x_i^* &\le \lfloor x_i^0 \rfloor \\ x_i^* &\ge \lfloor x_i^0 \rfloor + 1 \end{aligned}$$

must be true.

We now choose one of the subproblems, say Problem 1, which is after all an LP, and solve it. The solution x^1 will in general not be integer, and we may split Problem 1 into two subproblems just as we split Problem 0, creating Problems 3 and 4. We can visualize this process continuing indefinitely as a successively finer and finer subdivision of the feasible region, as shown in Fig. 18–2. Each subset in a given partition represents a subproblem i, with relaxed solution x^i and lower bound $z_i = c(x^i)$ on the cost of any solution in the partition.

We can also visualize this process as a tree, as shown in Fig. 18–3. The root represents the original feasible region and each node represents a subproblem.

al is not
optimal
integer,
w add a
tions to
roblems
ose that
lems are

(18.2)

(18.3)

2, 1) and
ution x^0
ems gen-
cing the

n of one

after all
l we may
ng Prob-
a succes-
ig. 18–2.
l solution
n.
The root
problem.

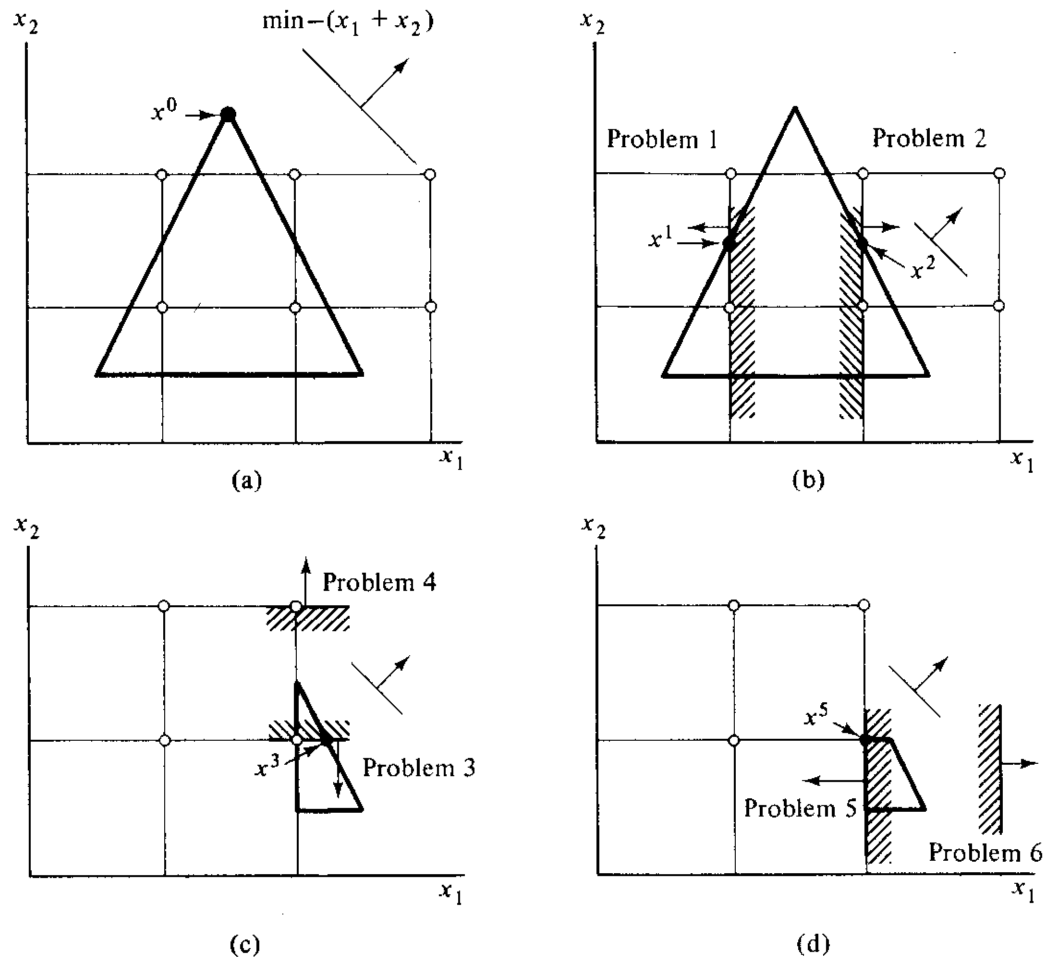


Figure 18–1 Stages in the solution of an ILP by branch-and-bound.

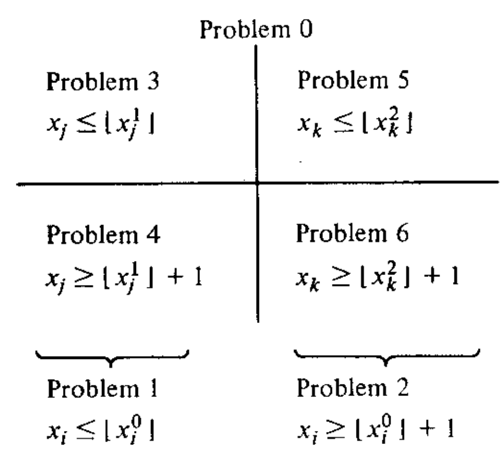


Figure 18–2 Successive subdivision of the feasible region of an ILP by addition of inequalities.

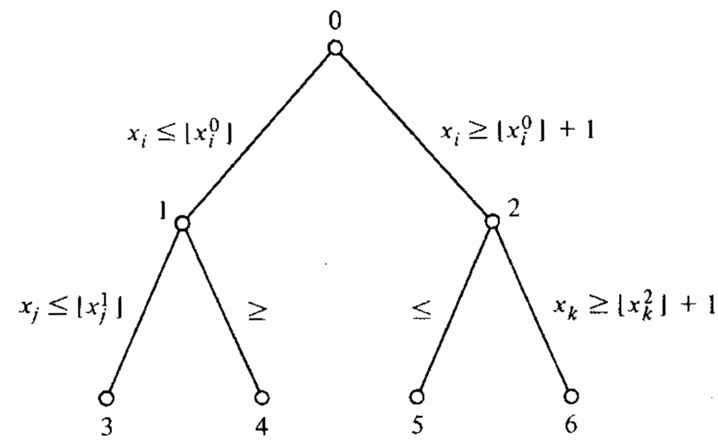


Figure 18–3 Representation of solution space subdivision by a binary tree.

Splitting the feasible region at a node by the addition of the inequalities Eqs. 18.2 and 18.3 is represented by the branching to the node's two children.

If the original ILP has a bounded feasible region, this process cannot continue indefinitely, because eventually the inequalities at a node in the branching tree will lead to an integer solution to the corresponding LP, which is an optimal solution to the original ILP (see Problem 1). The branching process can fail at a particular node for one of two reasons: (1) the LP solution can be integer; or (2) the LP problem can be infeasible.

Example 18.1 (Continued)

If we continue branching from Problem 2 in the example in Fig. 18–1, we obtain the branching tree shown in Fig. 18–4. Three leaves are reached in the right subtree; these leaves correspond to two infeasible LP's, and one LP with an integer solution $x^5 = (2, 1)$ with cost $z_5 = c(x^5) = -3$. □

What we have described up to this point comprises the *branching* part of branch-and-bound. If we continue the branching process until all the nodes are leaves of the tree and correspond either to integer solutions or infeasible LP's, then the leaf with the smallest cost must be the optimal solution to the original ILP. We come now to an important component of the branch-and-bound approach: Suppose at some point the *best complete integer solution* obtained so far has cost z_m and that we are contemplating branching from a node at which the lower bound $z_k = c(x^k)$ is *greater than or equal to* z_m. This means that any solution x that would be obtained as a descendent of x^k would have cost

$$c(x) \ge z_k \ge z_m$$

and hence we need not proceed with a branching from x^k. In such a case, we say that the node x^k has been *killed*, and refer to it (as one might guess) as *dead.*

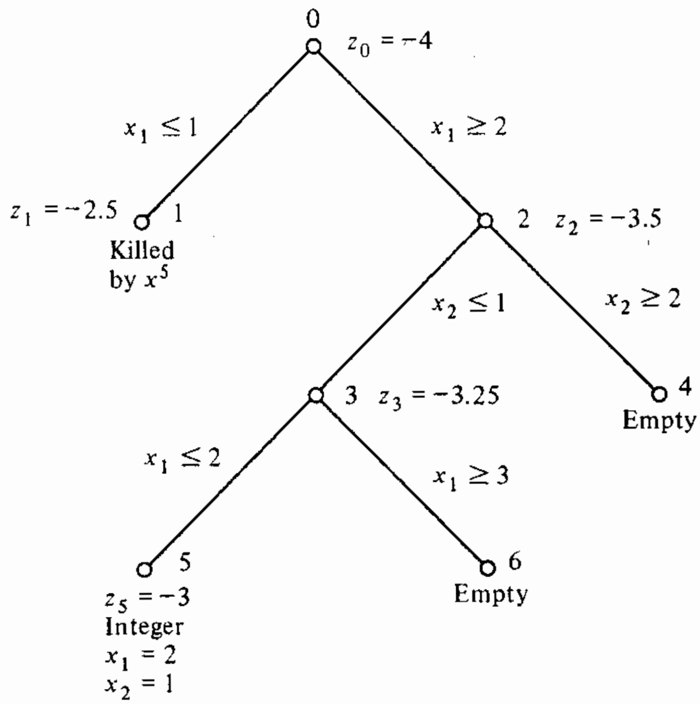


Figure 18–4 The binary tree leading to a solution to the problem.

(The term *fathomed* is also used.) The remaining nodes, from which branching is still possibly fruitful, are referred to as *live.*

Example 18.1 (Continued)

Referring again to Fig. 18–1 and 18–4, the node corresponding to Problem 1 has associated with it a lower bound of -2.5, which is greater than the solution cost of -3 associated with node 5. It is therefore killed by node 5, as shown. Since no live nodes remain, node 5 must represent the optimal solution. □

There are now still two important details in the algorithm that need to be specified: We must decide how to choose, at each branching step, which node to branch from; and we must decide how to choose which noninteger variable is to determine the added constraint. Dakin [Da] recommends branching in a depth-first manner to reduce the amount of storage needed for intermediate tableaux. If storage is not a determining factor, branching from the live node with the lowest lower bound might seem to be a reasonable heuristic. We shall discuss this problem later on in this chapter in a more general context; but little is known in general about this choice.

As for the second choice—the variable to add the constraint—Dakin [Da] reports that the best strategy is to find that constraint which leads to the largest increase in the lower bound z after performing one iteration of the dual simplex algorithm after that constraint is added, and to add either that constraint or its alternative [Da]. The motivation is to find the branch from a given node that is most likely to get killed, and in this way keep the search tree shallow. Again, there are no theorems to tell us the best strategy, and computational experience and intuition are the only guides to the design of fast algorithms of this type known at this time.

The idea of branch-and-bound is applicable not only to a problem formulated as an ILP (or mixed ILP), but to almost any problem of a combinatorial nature. We next develop the algorithm in a very general context.

18.2 Branch-and-Bound in a General Context

Two things were needed to develop the tree in the branch-and-bound algorithm for ILP.

1. *Branching* A set of solutions, which is represented by a node, can be partitioned into mutually exclusive sets. Each subset in the partition is represented by a child of the original node.
2. *Lower bounding* An algorithm is available for calculating a lower bound on the cost of any solution in a given subset.

No other properties of ILP were used. We may therefore formulate the method for any optimization problem in which (1) and (2) are available, whether or not the cost function or constraints are linear.

Figure 18–5 shows the basic algorithm. We use the set *activeset* to hold the live nodes at any point; the variable U is used to hold the cost of the best complete solution obtained at any given time (U is an *upper bound* on the optimal

```
begin
  activeset:={0}; (comment: "0" is the original problem)
  U:=∞;
  currentbest:=anything;
  while activeset is not empty do
    begin
      choose a branching node, node k ∈ activeset;
      remove node k from activeset;
      generate the children of node k, child i, i = 1, . . . , n_k,
      and the corresponding lower bounds, z_i;
      for i = 1, . . . ,n_k do
        begin
          if z_i ≥ U then kill child i
            else if child i is a complete solution then
                    U:=z_i, currentbest:=child i
                    else add child i to activeset
        end
    end
end
```

Figure 18–5 The basic branch-and-bound algorithm.

cost.) Notice that the branching process need not produce only two children of a given node, as in the ILP version, but any finite number.

Example 18.2 (The Shortest-Path Problem)

The shortest-path problem with nonnegative arc weights provides us with a transparent application of branch-and-bound, although we already have an efficient algorithm for its solution. Figure 18–6(a) shows an instance of the shortest-path problem. To solve this instance (Problem 0) by branch-and-bound we *branch* by choosing the next arc with which to continue the path. Thus a subset of feasible solutions corresponds to all paths from s to t that start by the choices already made. Fig. 18–6(b) shows a snapshot of the search tree that results when we branch from a node with the lowest lower bound at any point. The lower bound used is quite naturally the cumulative length of the partial path up to the particular point in the graph represented by the node in the search tree. For example, the path b-g-l brings us to a node with lower bound 10, the sum of the costs of the edges b, g, and l. Notice that in this example it is quite easy to compute at once the lower bounds for all the children of a node at which branching is taking place. In the ILP application, we needed to solve a linear program to obtain a lower bound, so we did not necessarily find both of the lower bounds immediately on branching, with the hope that we might avoid some of these calculations by killing later on.

Figure 18–6(c) shows the final search tree for this example. The first complete solution found has a cost of 8, and this turns out to be optimal. The killed nodes are indicated by bars below their lower bounds. □

Example 18.3 (The Traveling Salesman Problem)

A more realistic application of branch-and-bound is provided by an *NP*-complete problem like the TSP. There is more than one way to formulate a branching process for the TSP; the simplest, perhaps, is to partition the solution space into two sets at any point, according to whether a given edge is or is not in the tour. Little and others [LMSK] use this approach, together with a heuristic for the lower bound.

Another approach, attributed to Eastman [Ea], makes use of the fact that an efficient algorithm exists for the weighted bipartite matching, or assignment, problem (Chapter 11). This provides us with yet another example of one problem showing up as an easier subproblem of a more difficult one.

If we let the variable $x_{ij} = 1$ if edge $[i, j]$ is in a tour and zero otherwise, a tour for the n-city TSP with weights c_{ij} must satisfy

$$
\begin{aligned}
&\min z = \sum_{i,j=1}^{n} c_{ij}x_{ij} \\
&\sum_{i=1}^{n} x_{ij} = 1 \qquad\qquad j = 1, \ldots, n \qquad\qquad (18.4)\\
&\sum_{j=1}^{n} x_{ij} = 1 \qquad\qquad i = 1, \ldots, n
\end{aligned}
$$

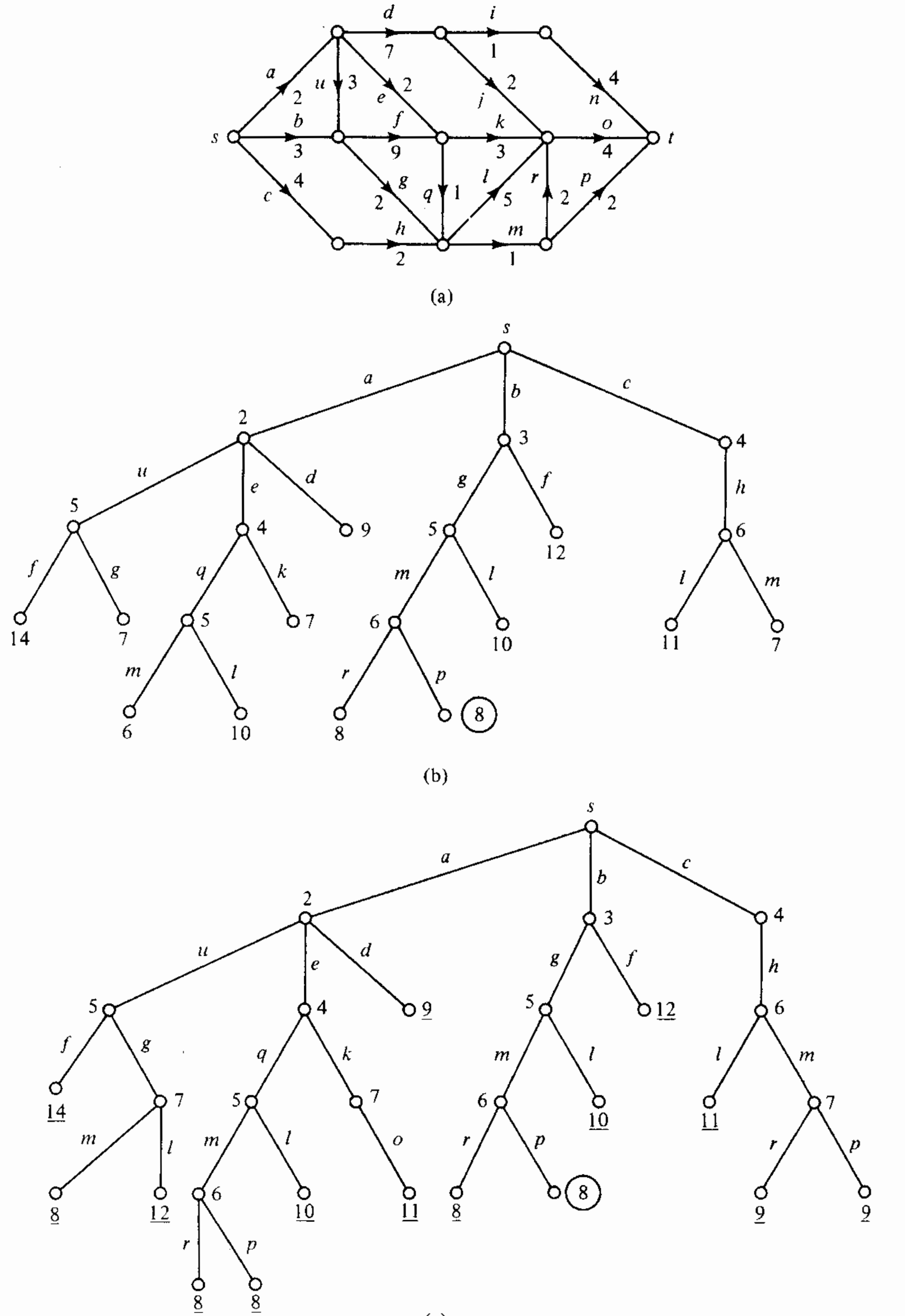


Figure 18–6 A shortest-path problem and its solution by branch-and-bound.

The two s
and leave
is not suff
ensuring
constraint
to (18.4) y
Furth
is not a tc
$[x_{12}, x_{23}, .$

Not all the
the solutio

x

at a time.
(Just put a
The branc
problem at
bound at t

Example 1

Held a
algorithm f
spanning tr

Definition

Given
nodes, a *1-t*
edges incide

The two sets of equality constraints express the fact that exactly one edge enters and leaves each node. This formulation (the assignment problem of Sec. 11.2) is not sufficient to capture the difficulty of the TSP, since we have no way of ensuring that the solution is a single tour with precisely one cycle. Thus the constraints in (18.4) are necessary but not sufficient for the TSP, and a solution to (18.4) yields a value of z that is a lower bound on the cost of the TSP.

Furthermore, if the solution to (18.4) *is* a tour, then it solves our TSP. If it is not a tour, then the solution contains a cycle of length less than n, a *subtour* $[x_{12}, x_{23}, \dots, x_{k1}]$ with

$$x_{12} = x_{23} = \dots = x_{k1} = 1$$

Not all these variables can equal one in a solution to the TSP, so we can partition the solution space into k subsets by adding one of the constraints

$$\begin{aligned} x_{12} &= 0 \\ x_{23} &= 0 \\ &\vdots \\ x_{k1} &= 0 \end{aligned}$$

at a time. This yields k problems, each of which is also an assignment problem. (Just put a very high cost on the edge x_{ij} we wish to exclude from the solution.) The branching step is illustrated in Fig. 18–7. The solution to the assignment problem at any node, by the methods of Chapter 11, then provides the lower bound at that node. □

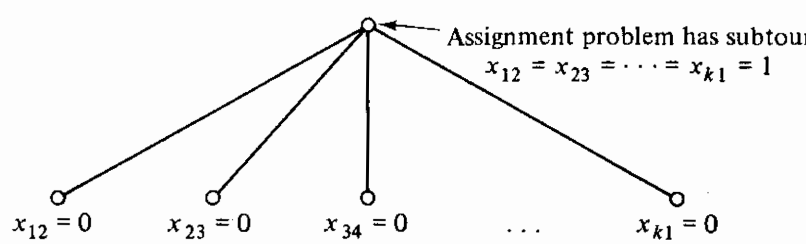


Figure 18–7 The branching step of branch-and-bound for the TSP.

Example 18.4 (A Spanning Tree Bound for the TSP)

Held and Karp [HK1, HK2] describe a very effective branch-and-bound algorithm for the TSP, based on a lower bound computed from related minimal spanning trees. We need the following idea.

Definition 18.1

Given a complete graph $G = (V, E)$ with distance matrix $[d_{ij}]$ and $n = |V|$ nodes, a *1-tree* is a graph formed by a tree on the node set $\{2, \dots, n\}$, plus two edges incident on node 1. □

Now every tour is a 1-tree (but not vice-versa), so the minimal cost of a 1-tree is a lower bound on the cost of a tour. Furthermore, the cost of a minimal 1-tree is easy to compute (see Problem 2). If we branch in a branch-and-bound algorithm for the TSP by including and excluding sets of edges, the subproblems are also TSP problems, and the corresponding 1-tree problems give us lower bounds. Held and Karp do not rest at this point, but pursue much tighter lower bounds based on this idea.

Suppose that we transform a TSP by replacing its distance matrix by $[d_{ij} + \pi_i + \pi_j]$ for some numbers π_i. Then every tour has its cost increased by the amount

$$2 \sum_{i=1}^{n} \pi_i$$

because every node is entered and exited exactly once. Thus the relative ranking of the costs of tours is unaffected by this transformation, and in particular an optimal tour remains optimal. On the other hand, the optimal 1-tree may change. If the degree of node i in a 1-tree is δ_i, then the cost of a 1-tree, after a transformation of the distance matrix by π, is

$$c + \sum_{i=1}^{n} \delta_i \pi_i$$

where c is the cost of the 1-tree with the original cost matrix $[d_{ij}]$. If c^* is the cost of an optimal tour, we therefore have

$$c^* + 2\sum_{i=1}^{n} \pi_i \geq \min_{\text{all 1-trees}} \left[c + \sum_{i=1}^{n} \delta_i \pi_i \right]$$

We can rewrite this as

$$c^* \geq w(\pi)$$

where

$$w(\pi) = \min_{\text{all 1-trees}} \left[c + \sum_{i=1}^{n} (\delta_i - 2)\pi_i \right]$$

This provides a lower bound $w(\pi)$ for any choice of π; Held and Karp pursue the problem of maximizing $w(\pi)$ with respect to π, thus obtaining the best lower bound possible with this idea. (It is worth mentioning that, in general, $c^* > \max_{\pi} w(\pi)$, so that there is a "gap" between the TSP and this corresponding lower-bounding problem.)

This bound obtained in [HK2] is so tight that the search trees for some fairly large problems (up to $n = 64$) can be exhibited in their entirety. This is a dramatic example of the importance of an effective lower bound in the branch-and-bound approach, since previous applications resulted in much larger search trees. □

18.3 Dominance Relations

Thus far, the only way a node can be eliminated for contention as the ancestor of an optimal solution—that is, killed—is by having its lower bound above the current upper bound. There is another way to kill a node, however: Consider the shortest-path problem in Example 18.2, for example. Suppose we branch to obtain the two nodes determined by edges a, e, q (with a lower bound of 5) and c, h (with a lower bound of 6). These two paths lead to the same point in the original graph, and we may say that the two paths in the branching tree have *merged.* There is no sense in pursuing the c, h path, because we have reached the same point with less cost via the a, e, q path. We may therefore kill the node corresponding to the c, h path, even before any complete solutions have been obtained. This is an example of a *dominance* relation; we say that the c, h node is *dominated* by the a, e, q node. One general way to define such a relation is as follows.

Definition 18.2

If we can show at any point that the best descendant of a node y is at least as good as the best descendant of node x, then we say y *dominates* x, and y can kill x. □

The existence of a practical algorithm for testing dominance depends very much on the particular problem at hand.

18.4 Branch-and-Bound Strategies

There are now many choices in how we implement a branch-and-bound algorithm for a given problem; we shall discuss some of them in this section.

First, there is the choice of the branching itself—there may be many schemes for partitioning the solution space, as in the TSP, or in the general ILP, for that matter.

Next, there is the lower-bound calculation. One often has a choice here between bounds that are relatively tight but require relatively large computation time and bounds that are not so tight but can be computed fast. A similar tradeoff may exist in the choice of a dominance relation.

Third, there are many ways in which we can use the lower-bound and dominance relations. To explain, let $AS(x)$ be the *activeset* when node x is branched from; let $CH(x)$ be the set of children of x in the branching tree; and let the upper bound $U(x)$ be the best cost of a complete solution when x is branched from (see Fig. 18–5.). Then Fig. 18–8 shows four possible ways in which nodes can be killed:

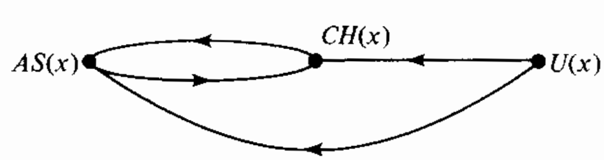


Figure 18–8 Possible ways in which nodes can be killed.

(a) A live node in $AS(x)$ can kill nodes in $CH(x)$.

(b) Nodes in $CH(x)$ can kill nodes in $AS(x)$.

(c), (d) The upper bound $U(x)$ can kill nodes in $AS(x)$ or $CH(x)$.

Again there is a trade-off in the choice of what is to be implemented: the time taken to test $CH(x)$ against $AS(x)$ may or may not be worthwhile in any particular problem.

Fourth, there is the choice at each branching step of which node to branch from. The usual alternatives are least-lower-bound-next, last-in-first-out, or first-in-first-out.

Still another choice must be made at the start of the algorithm. It is often practical to generate an initial solution by some heuristic construction, such as those described in Chapters 17 or 19. This gives us an initial upper bound $U < \infty$ and may be very useful for killing nodes early in the algorithm. As usual, however, we must trade off the time required for the heuristic against possible benefit.

Finally, we should mention that the branch-and-bound algorithm is often terminated before optimality is reached, either by design or necessity. In such a case we have a complete solution with cost U, and the lowest lower bound L of any live node provides a lower bound on the optimal cost. We are therefore within a ratio of $(U - L)/L$ of optimal.

It should be clear by now that the branch-and-bound idea is not one specific algorithm, but rather a very wide class. Its effective use is dependent on the design of a strategy for the particular problem at hand, and at this time is as much art as science. We conclude the discussion of branch-and-bound with an example of its application to a scheduling problem.

18.5 Application to a Flowshop Scheduling Problem

We now describe something of a "case history" of the application of branch-and-bound to a scheduling problem of some general interest—the two-machine flowshop scheduling problem with a sum-finishing-time criterion, defined as follows.

Definition 18.3

We are given a set of n jobs, J_i, $i = 1, \ldots, n$. Each job has two tasks, each to be performed on one of two machines. Job J_j requires a processing time τ_{ij} on machine i, and each task must complete processing on machine 1 before starting on machine 2. Let F_{ji} be the time at which job i finishes on machine j. The *sum finishing time* is defined to be the sum of the times that all the jobs finish processing on machine 2:

$$f = \sum_{i=1}^{n} F_{2i}$$

The sum-finishing-time problem (SFTP) is the problem of determining the order in which to assign the tasks to machines so f is minimum. □

Example 18.5

A common example of a situation in which a problem like SFTP may arise is a computer that executes one program at a time. We can identify the jobs with individual computer programs, machine 1 with the central processor, and machine 2 with the printer. We then assume that we are given a set of jobs with known execution and printing times and wish to schedule them so that the sum (or, equivalently, the average) finishing time is as small as possible. □

We now quote two important facts about SFTP. The first can be found in Conway, Maxwell, and Miller [CMM] and allows us to restrict our search for a single permutation that determines a complete schedule.

Theorem 18.1 *There is an optimal schedule for SFTP in which both machines process the jobs in the same order with no unnecessary idle time between jobs. (These are called* permutation *schedules.)*

The second, more recent, result is due to Garey, Johnson, and Sethi [GJS] and justifies the serious pursuit of a branch-and-bound algorithm.

Theorem 18.2 *The problem SFTP is NP-complete. (We mean, of course, the* yes-no *problem corresponding to SFTP with a solution of cost less than or equal to some L.)*

Example 18.6

Consider the following numerical example with 3 jobs.

τ_{ij}	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

Figure 18–9 shows all 6 possible permutation schedules, among which the optimal schedule must lie, by Theorem 18.1. The unique optimum has cost 18. □

```
Machine 1    1 1 2 2 2 3 3
Machine 2      1    2   3 3 3        f = 19
                ↑    ↑     ↑

Machine 1    1 1 3 3 2 2 2
Machine 2      1   3 3 3 2           f = 18
                ↑       ↑ ↑

Machine 1    2 2 2 1 1 3 3
Machine 2        2   1   3 3 3       f = 20
                  ↑   ↑     ↑

Machine 1    2 2 2 3 3 1 1
Machine 2        2   3 3 3 1         f = 21
                  ↑       ↑ ↑

Machine 1    3 3 1 1 2 2 2
Machine 2      3 3 3 1   2           f = 19
                    ↑ ↑   ↑

Machine 1    3 3 2 2 2 1 1
Machine 2      3 3 3 2   1           f = 19
                    ↑ ↑   ↑
```

Figure 18–9 The six possible permutation schedules in Example 18-5, and their associated costs. The arrows indicate finishing times on machine 2.

This problem is, with the help of Theorem 18.1, a problem of finding one permutation of n objects, and the natural way to branch is to choose the first job to be scheduled at the first level of the branching tree, the second job at the next level, and so on. What we need next is a lower-bound function.

Ignall and Schrage [IS] describe a very effective lower bound, which we derive here. Suppose we are at a node at which the jobs in the set $M \subseteq \{1, \ldots, n\}$ have been scheduled, where $|M| = r$. Let t_k, $k = 1, \ldots, n$, be the index of the kth job under any schedule which is a descendant of the node under consideration. The cost of this schedule, which we wish to bound, is

$$f = \sum_{i \in M} F_{2i} + \sum_{i \notin M} F_{2i} \tag{18.5}$$

Now if every job could start its processing on machine 2 immediately after completing its processing on machine 1, the second sum in Eq. 18.5 would become

$$S_1 = \sum_{k=r+1}^{n} [F_{1t_r} + (n - k + 1)\tau_{1t_k} + \tau_{2t_k}] \tag{18.6}$$

(see Problem 3). If that is not possible, S_1 can only increase, so

$$\sum_{i \notin M} F_{2i} \geq S_1 \tag{18.7}$$

Similarly, if every job can start on machine 2 immediately after the preceding job finishes on machine 2, the second sum in Eq. 18.5 would become

$$S_2 = \sum_{k=r+1}^{n} [\max (F_{2t_r}, F_{1t_r} + \min_{i \notin M} \tau_{1i}) + (n - k + 1)\tau_{2t_k}] \tag{18.8}$$

Again, this is a lower bound:

$$\sum_{i \notin M} F_{2i} \geq S_2 \tag{18.9}$$

Therefore

$$f \geq \sum_{i \in M} F_{2i} + \max(S_1, S_2) \tag{18.10}$$

The bound depends on the way the remaining jobs are scheduled, through t_k. This dependence can be eliminated by noting that S_1 is minimized by choosing t_k so that the tasks of length τ_{1t_k} are in ascending order, and that S_2 is minimized by choosing t_k so that the tasks of length τ_{2t_k} are likewise in ascending order. Call the resulting minimum values $\hat{S}_1$ and $\hat{S}_2$. Then

$$f \geq \sum_{i \in M} F_{2i} + \max(\hat{S}_1, \hat{S}_2) \tag{18.11}$$

is an easily computed lower bound.

Example 18.6 (Continued)

At the first branching step, Eq. 18.11 yields the lower bounds

$$f = \begin{cases} 18 & \text{if job 1 is scheduled first} \\ 20 & \text{if job 2 is scheduled first} \\ 18 & \text{if job 3 is scheduled first} \end{cases}$$

From the results in Figure 18–9, we see that the first two of these are as low as possible. Figure 18–10 shows a complete search tree in which the least lower bound is branched from first, from left to right in case of ties. The optimal solution (1, 3, 2) kills all the others when it is obtained. □

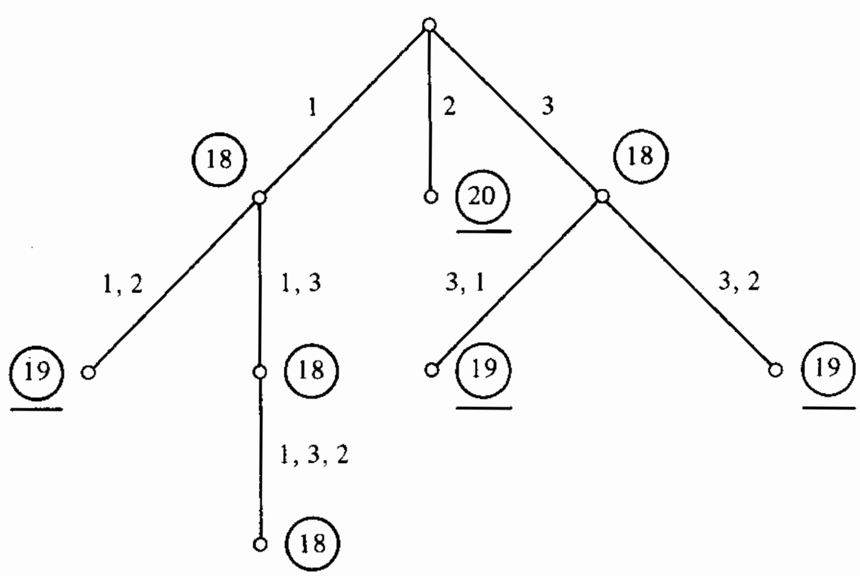


Figure 18–10 The search tree for Example 18.6.

Finally, we describe a natural dominance relation also given by Ignall and Schrage [IS]. Suppose we have two nodes t and u representing partial assignments of the same set of jobs, M. Let the kth scheduled job be t_k and u_k, $k = 1, \ldots, r$, under partial schedules t and u, respectively. Then if

$$F_{2t_r} \leq F_{2u_r} \tag{18.12}$$

(the set of jobs in M finishes no later on machine 2 under the partial schedule t), and if the accumulated cost under partial schedule t is no more than that under u,

$$\sum_{i \in M} F_{2i}\big|_{\text{schedule } t} \leq \sum_{i \in M} F_{2i}\big|_{\text{schedule } u} \tag{18.13}$$

then the best completion of schedule t is at least as good as the best completion of u. Equations 18.12 and 18.13 therefore define a dominance relation of t over u.

Example 18.6 (Continued)

Consider the nodes $t = (1, 2)$ and $u = (2, 1)$ (not generated in Fig. 18–10). Then t dominates u. On the other hand, $t = (1, 3)$ does not dominate $u = (3, 1)$. This instance is too small to show the real power of a dominance relation. □

18.6 Dynamic Programming

Dynamic programming is related to branch-and-bound in the sense that it performs an intelligent enumeration of all the feasible points of a problem, but it does so in a different way. The idea is to work backwards from the last decisions to the earlier ones.

Suppose we need to make a sequence of n decisions to solve a combinatorial optimization problem, say $D_1, D_2, \ldots, D_n$. Then if the sequence is optimal, the last k decisions, $D_{n-k+1}, D_{n-k+2}, \ldots, D_n$, must be optimal. That is, the *completion of an optimal sequence of decisions must be optimal.* This is often referred to as the *principle of optimality.*

The usual application of dynamic programming entails breaking down the problem into stages at which the decisions take place and finding a recurrence relation that takes us backward from one stage to the previous stage. We shall explain the method by example, starting with the shortest-path problem for layered networks, in which the sequence of decisions from last to first is clear.

Example 18.7 (Dynamic Programming for Shortest Path in Layered Networks)

Consider the layered network shown in Fig. 18.11, where we want to find the shortest s-t path. Let *table*(i) be a table of the optimal way to continue a shortest path when we are in the ith layer from the terminal t; that is, *table*(i) contains the best decision for each node in the layer i arcs from t. Thus the first

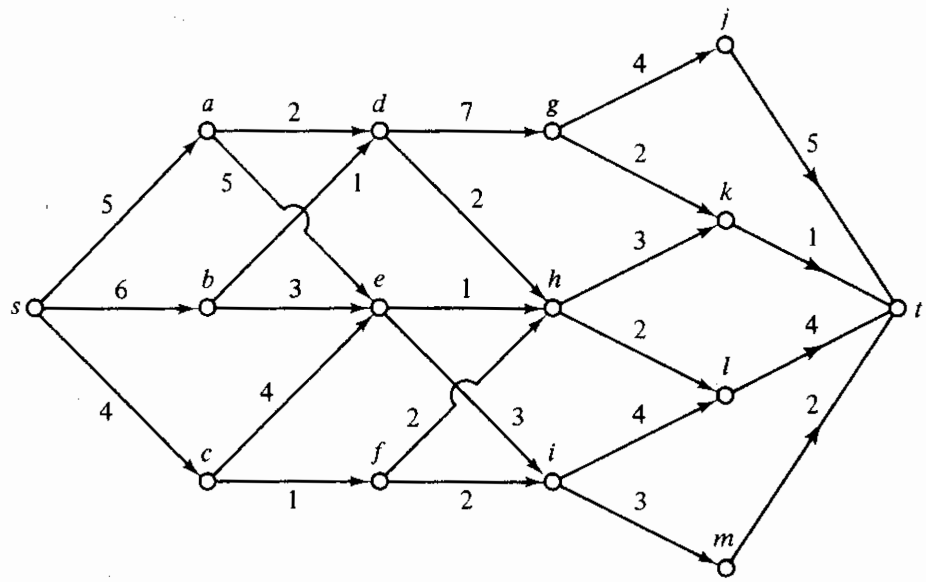


Figure 18–11 A shortest-path problem in a layered network.

table is simply

$$table(1) = \begin{cases} \text{node} & j \quad k \quad l \quad m \\ \text{next node} & t \quad t \quad t \quad t \\ \text{total cost} & 5 \quad 1 \quad 4 \quad 2 \end{cases}$$

Now consider the construction of *table*(2). At node g we can reach j or k. We know the cost of the optimal completion from j or k by *table*(1) and thus can find the best decision at node g by comparing the cost of arc (g, j) plus the cost of completion from j, with the cost of arc (g, k) plus the cost of completion from k. Continuing for nodes h and i, *table*(2) is

$$table(2) = \begin{cases} \text{node} & g \quad h \quad i \\ \text{next node} & k \quad k \quad m \\ \text{total cost} & 3 \quad 4 \quad 5 \end{cases}$$

Next we find

$$table(3) = \begin{cases} \text{node} & d \quad e \quad f \\ \text{next node} & h \quad h \quad h \\ \text{total cost} & 6 \quad 5 \quad 6 \end{cases}$$

$$table(4) = \begin{cases} \text{node} & a \quad b \quad c \\ \text{next node} & d \quad d \quad f \\ \text{total cost} & 8 \quad 7 \quad 7 \end{cases}$$

$$table(5) = \begin{cases} \text{node} & s \\ \text{next node} & c \\ \text{total cost} & 11 \end{cases}$$

We can now reconstruct an optimal path by backtracking through the tables, obtaining the path (s, c, f, h, k, t), with a cost of 11. ☐

Of course, in more difficult problems, dynamic programming runs into time and space problems, because the tables may grow in size at an exponential rate from stage to stage. To illustrate this, we conclude with a dynamic programming formulation of the TSP.

Example 18.8 (Dynamic Programming and the TSP [HK3])

Given a set $S \subseteq \{2, 3, \ldots, n\}$ and $k \in S$, we let $C(S, k)$ be the optimal cost of starting from city 1, visiting all the cities in S, and ending at city k. We begin by finding $C(S, k)$ for $|S| = 1$, which is simply

$$C(\{k\}, k) = d_{1k} \qquad \text{all } k = 2, \ldots, n \tag{18.14}$$

To calculate $C(S, k)$ for $|S| > 1$, we argue that the best way to accomplish our journey from 1 to all of S, ending at k, is to consider visiting m immediately before k, for all m, and looking up $C(S - \{k\}, m\}$ in our preceding table. Thus

$$C(S, k) = \min_{m \in S - \{k\}} [C(S - \{k\}, m) + d_{mk}] \tag{18.15}$$

This must be calculated for all sets S of a given size and for each possible city m in S. (We also must save the city m for which a minimum is achieved, so that we can reconstruct the optimal tour by backtracking.) If we count each value of $C(S, k)$ as one storage location, we need space equal to

$$\sum_{k=1}^{n-1} k\binom{n-1}{k} = (n-1)2^{n-2} = O(n2^n) \tag{18.16}$$

locations [HK3] and a number of additions and comparisons equal to

$$\sum_{k=2}^{n-1} k(k-1)\binom{n-1}{k} + (n-1) = (n-1)(n-2)2^{n-3} + (n-1) = O(n^2 2^n) \tag{18.17}$$

These are exponential functions of the problem size n, and may seem prohibitively large. But when we consider the fact that there are $(n - 1)!$ distinct tours in a naïve enumeration, we see that in fact this approach results in enormous savings. Since there is no algorithm known for the TSP that is better than exponential, the dynamic programming approach cannot be dismissed out of hand, although branch-and-bound algorithms have proven more effective in this application. ☐

As can be seen from the two examples above, dynamic programming is a very general idea and can demand varying amounts of ingenuity to find good ways of breaking down a problem into stages so that a convenient recurrence relation can be found. Some thought will show that some of the algorithms seen earlier in this book can be considered to be applications of dynamic programming (see Problem 7 and 8 and Section 17.3 on the 0-1 KNAPSACK problem).

PROBLEMS

1. Prove that the branch-and-bound algorithm when applied to ILP terminates at optimality within a number of steps bounded by an exponential in the problem size.

2. Describe an $O(n^2)$-time algorithm for finding a minimum 1-tree, given an $n \times n$ distance matrix.

3. Prove that Eqs. 18.6 and 18.8 are the claimed lower bounds.

4. Analyze the time and space complexities of the dynamic programming algorithm for shortest path in a layered network and compare them with the time and space complexities of Dijkstra's algorithm (Chapter 6) applied to the same problem.

5. Establish Eqs. 18.16 and 18.17, giving the space and time requirements of the dynamic programming algorithm for the TSP. What is the asymptotic savings in time over complete enumeration of $(n - 1)!$ tours?

6. In the branching step of the general branch-and-bound algorithm, is it necessary that the partition of the set of solutions be a *disjoint* partition?

7. Interpret Dijkstra's algorithm (Chapter 6) for shortest path (with nonnegative distances) as an application of dynamic programming. Define explicitly the definition of a stage and the recurrence relation and boundary conditions analogous to Eqs. 18.14 and 18.15.

8. Repeat Problem 7 for the Floyd-Warshall algorithm (See. 6.5).

9. We shall derive an $O(|V|^3)$ algorithm for shortest path with negative distances allowed, using dynamic programming. Consider an undirected graph $G = (V, E)$ with source node s and distance matrix $[d_{ij}]$.

   (a) Let the label $p_i(x)$ be the shortest length of any path from source s to node x, using i or fewer intermediate edges. Write the recurrence relation for $p_i(x)$ and its boundary conditions.

   (b) Show that if no $p_i(x)$ changes from stage i to $i + 1$, we have reached optimality and can stop.

   (c) Show that if we have not converged in the sense of Part (b) after $i = |V|$ stages, there is a negative-cost cycle. From this, prove that the algorithm takes $O(|V|^3)$ time.

   (d) Compare the worst-case behavior with that of the Floyd-Warshall algorithm.

10. Suppose we want to compute the product of n matrices, $A_1, A_2, \ldots, A_n$, where A_i has p_i rows and q_i columns. We assume, of course, that they are compatible;

that is, $q_i = p_{i+1}$, $i = 1, \ldots, n-1$. Devise an $O(n^3)$-time dynamic programming algorithm to find the order in which to multiply these matrices to minimize the total number of scalar multiplications, assuming that multiplying A_i and A_{i+1} takes $p_i q_i q_{i+1}$ scalar multiplications. How much space does your algorithm require?

11. Explain why Algorithm DP-I in Sec. 17.3 for 0-1 KNAPSACK is considered a dynamic programming algorithm.

12. Reformulate the dynamic programming algorithm for the TSP (Example 18.8) as a branch-and-bound algorithm with no upper and lower bounds, but with a dominance relation.

NOTES AND REFERENCES

A good review of branch-and-bound, up to 1966, with pertinent early references, is

[LW] LAWLER, E. L., and D. E. WOOD, "Branch-and-Bound Methods: A Survey," *OR*, 14 (1966), 699–719.

Lawler and Wood describe the approach of Little and others to the TSP

[LMSK] LITTLE, J. D. C., K. G. MURTY, D. W. SWEENY, and C. KAREL, "An Algorithm for the Traveling-Salesman Problem," *OR*, 11 (1963), 972–89

as well as Eastman's:

[Ea] EASTMAN, W. L., "Linear Programming with Pattern Constraints," Ph.D. Thesis, Report No. BL. 20, The Computation Laboratory, Harvard University, 1958.

The formulation of ILP in Sec. 18.1 is given in

[Da] DAKIN, R. J., "A Tree-Search Algorithm for Mixed Integer Programming Problems," *Comp. J.*, 8, no. 3 (1965), 250–55.

The formulation of branch-and-bound given here, as well as the flowshop examples, are after W. H. Kohler and K. Steiglitz in Chapter 6 of

[Co] COFFMAN, E. G., JR., ed., *Computer and Job-Shop Scheduling*. New York: Wiley-Interscience, 1976.

The results of some computational experiments with branch-and-bound can be found there, as well as a dynamic programming formulation of a scheduling problem that turns out to be equivalent to branch-and-bound.

The lower bound and dominance relation for the flowshop problem is from

[IS] IGNALL, E., and L. SCHRAGE, "Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems," *OR*, 13, no. 3 (1965), 400–12.

Theorem 18.1 can be found in

[CMM] CONWAY, R. W., W. L. MAXWELL, and L. W. MILLER, *Theory of Scheduling*. Reading, Mass.: Addison-Wesley Publishing Co., Inc., 1967.

Theorem 18.2 is due to

[GJS] GAREY, M. R., D. S. JOHNSON, and R. SETHI, "The Complexity of Flowshop and Jobshop Scheduling," *Math. of Operations Res.*, 1 (1976), 117–29.

Held and Karp's work on 1-trees is from

[HK1] HELD, M., and R. M. KARP, "The Traveling Salesman Problem and Minimum Spanning Trees," *OR*, 18 (1970), 1138–62.

[HK2] ———, "The Traveling Salesman Problem and Minimum Spanning Trees: Part II," *Math. Prog.*, 1 (1971), 6–25.

The dynamic programming formulation of the TSP can be found in

[HK3] HELD, M., and R. M. KARP, "A Dynamic Programming Approach to Sequencing Problems," *J. SIAM*, 10, 1 (1962), 196–210.

as well as formulations of other combinatorial problems. Solutions to Problems 7–9 can be found in

[DL] DREYFUS, S. E., and A. M. LAW, *The Art and Theory of Dynamic Programming*. New York: Academic Press, Inc., 1977.

This book contains a comprehensive chapter on the shortest-path problem including some (constant-factor) improvements for the single-source and all-pairs problems. The algorithm in Problem 9 has an uncertain origin. In

[Dr] DREYFUS, S. E., "An Appraisal of Some Shortest-Path Algorithms," *OR*, 17 (1969), 395–412.

references to L. R. Ford, Jr., E. F. Moore, and R. E. Bellman are given, in 1956, 1957, and 1958, respectively.

NOTES AND REFERENCES

Section 19.2

[Bo] Bock, F., "An Algorithm for Solving 'Traveling-Salesman' and Related Network Optimization Problems," presented at the fourteenth National Meeting of the Op. Res. Soc. of America, St. Louis, Missouri (Oct. 24, 1958).

[Cr] Croes, G. A., "A Method for Solving Traveling-Salesman Problems," *OR*, 6, no. 6 (November–December 1958), 791–812.

[Li] Lin, S., "Computer Solutions of the Traveling Salesman Problem," *BSTJ*, 44, no. 10 (December 1965), 2245–69.

[HK1] Held, M., and R. M. Karp, "A Dynamic Programming Approach to Sequencing Problems," *J. SIAM*, 10, no. 1 (March 1962), 196–210.

An early description of local search can be found in

[DFFN] Dunham, B., D. Fridshal, R. Fridshal, and J. H. North, "Design by Natural Selection," IBM Res. Dept. RC-476, June 20, 1961.

Section 19.3

[StWK] Steiglitz, K., P. Weiner and D. J. Kleitman, "The Design of Minimal Cost Survivable Networks," *IEEE Trans. Cir. Theory*, CT–16, no. 4 (1969), 455–60.

Section 19.4

[RFRSK] Rothfarb, B., H. Frank, D. M. Rosenbaum, K. Steiglitz, and D. J. Kleitman, "Optimal Design of Offshore Natural-Gas Pipeline Systems," *OR*, 18, no. 6 (November–December 1970), 992–1020.

Section 19.5

[KL] Kernighan, B. W., and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *BSTJ*, 49, no. 2 (February 1970), 291–307.

[LK] Lin, S., and B. W. Kernighan, "An Effective Heuristic for the Traveling-Salesman Problem," *OR*, 21 (1973), 498–516.

Section 19.6

[Gr] Gratzer, F. J., "Computer Solution of Large Multicommodity Flow Problems," (unpublished Ph.D. dissertation, Princeton University, September 1970).

[GS] Gratzer, F. J., and K. Steiglitz, "A Heuristic Approach to Large Multicommodity Flow Problems," *Proc. Symp. Computer-Communications Networks and Teletraffic*, Microwave Res. Inst. Symp. Series, Vol. XXII, Polytechnic Press, N.Y. (1972), pp. 311–24.

[Kr] KRONE, M. J., "Heuristic Programming Applied to Scheduling Problems," (unpublished Ph.D. dissertation, Princeton University, September 1970).

[KS] KRONE, M. J., and K. STEIGLITZ, "Heuristic-Programming Solution of a Flowshop-Scheduling Problem," *OR*, 22, no. 3 (May-June 1974), 629–38.

[GL] GOLDSTEIN, A. J., and A. B. LESK, "Common Feature Techniques for Discrete Optimization," Comp. Sci. Tech. Report 27, Bell Tel. Labs. (March 1975).

[SW] STEIGLITZ, K., and P. WEINER, "Some Improved Algorithms for Computer Solution of the Travelling Salesman Problem," *Proc. Sixth Allerton Conf. on Circuit and System Theory,* Urbana, Illinois (1968), 814–21.

Section 19.7

[Ro] ROCKAFELLAR, R. T., *Convex Analysis*. Princeton, N.J.: Princeton Univ. Press, 1970.

[Pa2] PAPADIMITRIOU, C. H., "The Adjacency Relation on the Traveling Salesman Polytope is *NP*-Complete," *Math. Prog.*, 14 (1978), 312–24.

[Sa1] SAVAGE, S. L., "The Solution of Discrete Linear Optimization Problems by Neighborhood Search Techniques," (unpublished Ph.D. dissertation, Yale University, June 1973).

[Sa2] SAVAGE, S. L., "Some Theoretical Implications of Local Search," *Math. Prog.*, 10 (1976), 354–66.

[WSB] WEINER, P., S. L. SAVAGE, and A. BAGCHI, "Neighborhood Search Algorithms for Guaranteeing Optimal Traveling Salesman Tours Must be Inefficient," *J. Comp. Sys. Sci.*, 12 (1976), 25–35.

[SaWK] SAVAGE, S. L., P. WEINER, and M. J. KRONE, "Convergent Local Search," Res. Report 14, Dept. Comp. Sci., Yale University, New Haven, Conn. (1973).

[KP] KARP, R. M., and C. H. PAPADIMITRIOU, "On Linear Characterizations of Combinatorial Optimization Problems," *Proc. Twenty-First Annual Symp. on Foundations of Comp. Sci.*, IEEE (1980), 1–9.

Section 19.8

[Pa1] PAPADIMITRIOU, C. H., "The Complexity of Combinatorial Optimization Problems," (unpublished Ph.D. Thesis, Princeton University, August 1976).

[PS1] PAPADIMITRIOU, C. H., and K. STEIGLITZ, "On the Complexity of Local Search for the Traveling Salesman Problem," *J. SIAM Comp.*, 6, 1 (March 1977), 76–83.

FURTHER NOTES AND REFERENCES

Applications of local search to other problems have been described in the literature. The following references report favorable results in designing mechanical linkages, such as the printing-bed drive of a printing-press mechanism:

[LF] LEE, T. W., and F. FREUDENSTEIN, "Heuristic Combinatorial Optimization in the Kinematic Design of Mechanisms, Part I: Theory, Part 2: Applications," *Journal of Engineering for Industry*, 98, no. 4 (November 1976), 1277–84.

[LL] LEE, T. W., and N. LANGRANA, "Heuristic Approaches to the Inversed Dynamic Linkage Problems: With Special Application to Biomechanics," *Proc. Fifth World Congress on Theory of Machines and Mechanisms*, Paper No. USA-53, Montreal, Canada (July 1979).

The design of small-diameter networks, the subject of Problem 7, is described in

[TS] TOUEG, S., and K. STEIGLITZ, "The Design of Small-Diameter Networks by Local Search," *IEEE Trans. on Computers*, C-28, no. 7 (July 1979), 537–42.

Problems 8 and 9 are from

[PS2] PAPADIMITRIOU, C. H., and K. STEIGLITZ, "Some Examples of Difficult Traveling Salesman Problems," *OR*, 26, no. 3 (May–June 1978), 434–43.

The uniform graph partitioning problem in Problem 1 is a variation of minimum cut into bounded sets; for a proof of *NP*-completeness see

[GJS] GAREY, M. R., D. S. JOHNSON, and L. J. STOCKMEYER, "Some Simplified *NP*-Complete Graph Problems," *Theor. Comp. Sci.*, 1 (1976), 237–67.

Problem 16(b) is based on a graph-theoretic result first conjectured by Lin [Li] and proved in

[TH] THOMASON, A. G., "Hamiltonian Cycles and Uniquely Edge Colourable Graphs," *Annals of Discrete Math.*, 3 (1978), 259–68.