13

The Cutting-Stock Problem

Materials such as paper, textiles, cellophane, and metallic foil are manufactured in rolls of large widths. These rolls, referred to as *raws*, are later cut into rolls of small widths, called *finals*. Each manufacturer produces raws of a few standard widths; the widths of the finals are specified by different customers and may vary widely. The cutting is done on machines by knives that slice through the rolls in much the same way as a knife slices bread. For example, a raw that is 100 in. wide may be cut into two finals with 31-in. widths and one final with a 36-in. width, with the 2 in. left over going to waste. When a complicated summary of orders has to be filled, the most economical way of cutting the existing raws into the desired finals is rarely obvious. The problem of finding such a way is known as the *cutting-stock problem*.

THE CUTTING-STOCK PROBLEM AND LINEAR PROGRAMMING

Let us consider an example where the raws are 100 in. wide and we have to fill the following summary of orders:

97 finals of width 45 in.

610 finals of width 36 in.

395 finals of width 31 in.

211 finals of width 14 in.

First, we shall tabulate all the ways of cutting a raw into a_1 finals of width 45, a_2 finals of width 36, a_3 finals of width 31, and a_4 finals of width 14. As it turns out, there are 37 ways of doing this. They are listed in Table 13.1, with the jth pattern specified by a_{1j}, a_{2j}, a_{3j}, and a_{4j}. Now we may speculate about various ways of filling the order summary. If the jth pattern is used x_j times then we must have

$$\sum_{j=1}^{37} a_{ij}x_j = b_i \qquad (i = 1, 2, 3, 4) \tag{13.1}$$

with $b_1 = 97, \quad b_2 = 610, \quad b_3 = 395, \quad b_4 = 211$. Thus we are led to

$$\text{minimize} \quad \sum_{j=1}^{37} x_j \qquad \text{subject to} \quad (13.1) \quad \text{and} \quad x_j \geq 0 \qquad (j = 1, 2, \ldots, 37). \tag{13.2}$$

Note that there is an obvious catch: an optimal solution of (13.2) may not be integer-valued and yet, in order to describe a realistic plan, our numbers x_j must be nonnegative *integers*. Nevertheless, as K. Eisemann (1957) pointed out, analogues of (13.2) may sometimes describe the actual problem with a sufficient degree of realism. For example, rolls of expensive materials, such as silk, are *not* simply cut through. Instead, while a raw roll is being unwound, it is sliced lengthwise for a certain total length. This process may be stopped at any time and the knife setting altered; the very same raw roll continues to unwind, now being sliced to different widths. In such situations, it does make sense to speak of a cutting pattern used a fractional number of times. In a paper mill, however, each raw roll is either sliced completely or not sliced at all; hence, a working plan is described by *integers* x_j. Even in those situations, though, it is advantageous to solve analogues of (13.2): their optimal solutions point out a variety of working plans that are, at least, close to optimal. For instance, an optimal solution of (13.2) is

$$x_1^* = 48.5, \quad x_{10}^* = 105.5, \quad x_{12}^* = 100.75, \quad x_{13}^* = 197.5.$$

Rounding each x_j^* down to the nearest integer, we obtain a schedule using 450 raws to produce 96 finals of width 45 in., 607 finals of width 36 in., 394 finals of width 31 in., and 210 finals of width 14 in. The residual unsatisfied demand can be easily satisfied by as few as three extra raws. Thus we find a way of satisfying the total demand by using only 453 raws. Since the optimum value of (13.2) is 452.25, our integer-valued solution is clearly optimal.

The same line of attack applies in general. However, there are difficulties, which stem from two radically different sources:

Table 13.1 Thirty-Seven Cutting Patterns

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
a_{1j}	2	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a_{2j}	0	1	1	0	0	0	0	0	0	2	2	2	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a_{3j}	0	0	0	1	1	0	0	0	0	0	0	0	2	1	1	1	0	0	0	0	0	3	2	2	2	1	1	1	1	1	0	0	0	0	0	0	0
a_{4j}	0	1	0	1	0	3	2	1	0	2	1	0	0	2	1	0	4	3	2	1	0	0	2	1	0	4	3	2	1	0	7	6	5	4	3	2	1

Note: If the width of the trimmed-off piece exceeds 14 in., which is the width of the smallest final, then the pattern is clearly wasteful. For instance, it would be foolish to use pattern 9 twice in a row: the resulting two 45-in. finals could just as well be cut from a *single* raw by pattern 1. These wasteful patterns, however, might be called for to take care of various odds and ends. In any event, as we shall see later, the cutting-stock problem is customarily solved *without* tabulating all the patterns in advance.

(i) Problems commonly encountered in the paper industry may involve astronomical numbers of variables. For example, if the raw rolls are 200 in. wide and if the finals are ordered in 40 different lengths ranging from 20 in. to 80 in., then the number of different patterns can easily exceed 10 or even 100 million. The amount of time and space required just to tabulate these patterns (not to mention *solving* the problem afterwards) may be simply unavailable.

(ii) Passing from an optimal *fractional-valued* solution to an optimal *integer-valued* solution is not easy. Rounding the fractional values down and then satisfying the residual demand, as we did in the example, may not yield the optimal working plan. If the finals are ordered in small enough quantities, then the patterns used in the optimal integer-valued solution may be quite different from those used originally in the optimal fractional-valued solutions. In fact, if the order summary is small, then experienced schedulers often find a working plan that is better than any plan obtained by simple adjustments of the fractional optimum; see R. E. D. Woolsey (1972a).

An ingenious way of getting around the first difficulty has been suggested by P. C. Gilmore and R. E. Gomory (1961). The trick, in short, is to work with only a few patterns at a time and to generate new patterns only when they are really needed. We are going to discuss this *delayed column-generation technique* in the next section. (Actually, delayed column generation was used even before Gilmore and Gomory in a different context; it constitutes a crucial part of the Dantzig–Wolfe decomposition algorithm presented in Chapter 26.)

No efficient way of handling the second difficulty is known. Fortunately, rounding the fractional optimum up or down, with subsequent ad hoc adjustments, is quite satisfactory for typical problems arising in the paper industry. If m different lengths of finals are ordered, then the fractional optimum produced by the simplex method involves at most m nonzero variables x_j. Hence the cost difference between the rounded-off solution and the true integer optimum will be *at most* the cost of m raws (and often considerably less). Since a typical value of m is no more than 50, and most final widths are ordered in hundreds or thousands, the possible cost increases due to inefficient rounding are relatively negligible. Problems with wide spectra of final widths are usually referred to as *bin-packing problems*; the term *cutting-stock problem* is normally reserved for bin-packing problems with narrow spectra of final widths and large order summaries. These characteristic features make cutting-stock problems amenable to the linear programming approach described here.

DELAYED COLUMN GENERATION

Consider a cutting-stock problem where the raws are r inches wide and the order summary calls for b_i finals of width w_i ($i = 1, 2, \ldots, m$). As in the previous example, we are led to

$$\text{minimize} \quad \mathbf{cx} \qquad \text{subject to} \quad \mathbf{Ax} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0}.$$

Here $\mathbf{b}$ is a column vector with components $b_1, b_2, \ldots, b_m$ and $\mathbf{c}$ is a row vector with components $1, 1, \ldots, 1$. Each column $\mathbf{a} = [a_1, a_2, \ldots, a_m]^T$ of $\mathbf{A}$ specifies a pattern cutting a raw into a_i finals of width w_i ($i = 1, 2, \ldots, m$). Thus $\mathbf{a}$ is a column of $\mathbf{A}$ if and only if $a_1, a_2, \ldots, a_m$ are nonnegative integers such that $\sum w_i a_i \leq r$. The revised simplex method refers to the nonbasic columns of $\mathbf{A}$ only in step 2 of each iteration, when a new entering column $\mathbf{a}$ has to be found; having computed the row vector $\mathbf{y}$, we look for nonnegative integers $a_1, a_2, \ldots, a_m$ such that

$$\sum_{i=1}^{m} w_i a_i \leq r \quad \text{and} \quad \sum_{i=1}^{m} y_i a_i > 1 \tag{13.3}$$

(we are *minimizing* the objective function). If we can always find such integers or prove their nonexistence, then there is no need to tabulate all the columns of **A** in advance. Instead, the entering columns may be generated individually at the rate of one column per iteration.

To illustrate the details, we shall consider an example where the raws are 91 in. wide and the order summary calls for

78 finals of width $25\frac{1}{2}$ in.

40 finals of width $22\frac{1}{2}$ in.

30 finals of width 20 in.

30 finals of width 15 in.

Here, the revised simplex method may be initialized by

$$\mathbf{B} = \begin{bmatrix} 3 & & & \\ & 4 & & \\ & & 4 & \\ & & & 6 \end{bmatrix} \quad \text{and} \quad \mathbf{x}_B^* = \begin{bmatrix} 26 \\ 10 \\ 7.5 \\ 5 \end{bmatrix}.$$

In general, one can always initialize by m cutting patterns such that the ith pattern yields only finals of width w_i. Now the first iteration may begin.

Step 1. Solving the system $\mathbf{yB} = [1, 1, 1, 1]$, we obtain $\mathbf{y} = [\frac{1}{3}, \frac{1}{4}, \frac{1}{4}, \frac{1}{6}]$.

Step 2. Now we are looking for nonnegative integers a_1, a_2, a_3, a_4 such that

$$25.5a_1 + 22.5a_2 + 20a_3 + 15a_4 \leq 91$$

$$\frac{1}{3}a_1 + \frac{1}{4}a_2 + \frac{1}{4}a_3 + \frac{1}{6}a_4 > 1\ .$$

A certain method, which will be discussed in the next section, finds $a_1 = 2, a_2 = 0$, $a_3 = 2, a_4 = 0$. Hence $\mathbf{a} = [2, 0, 2, 0]^T$ is our entering column.

Step 3. Solving the system $\mathbf{Bd} = \mathbf{a}$, we find $\mathbf{d} = [\frac{2}{3}, 0, \frac{1}{2}, 0]^T$.

Step 4. Comparing the ratios $26/\frac{2}{3}$ and $7.5/\frac{1}{2}$, we find that $t = 15$ and that the third column has to leave **B**.

Step 5. Now we have

$$\mathbf{B} = \begin{bmatrix} 3 & & 2 & \\ & 4 & & \\ & & 2 & \\ & & & 6 \end{bmatrix} \quad \text{and} \quad \mathbf{x}_B^* = \begin{bmatrix} 26 - 2t/3 \\ 10 \\ t \\ 5 \end{bmatrix} = \begin{bmatrix} 16 \\ 10 \\ 15 \\ 5 \end{bmatrix}.$$

The second iteration begins.

Step 2. Now we are looking for nonnegative integers a_1, a_2, a_3, a_4 such that

$$25.5a_1 + 22.5a_2 + 20a_3 + 15a_4 \leq 91$$

$$\frac{1}{3}a_1 + \frac{1}{4}a_2 + \frac{1}{6}a_3 + \frac{1}{6}a_4 > 1 .$$

The method to be discussed later finds $a_1 = 2$, $a_2 = 1$, $a_3 = 0$, $a_4 = 1$. Hence, $\mathbf{a} = [2, 1, 0, 1]^T$ is our entering column.

Step 3. Solving the system $\mathbf{Bd} = \mathbf{a}$, we find $\mathbf{d} = [\frac{2}{3}, \frac{1}{4}, 0, \frac{1}{6}]^T$.

Step 4. Comparing the ratios $16/\frac{2}{3}$, $10/\frac{1}{4}$, and $5/\frac{1}{6}$, we find that $t = 24$ and that the first column has to leave $\mathbf{B}$.

Step 5. Now we have

$$\mathbf{B} = \begin{bmatrix} 2 & & 2 & \\ 1 & 4 & & \\ & & 2 & \\ 1 & & & 6 \end{bmatrix} \quad \text{and} \quad \mathbf{x}_B^* = \begin{bmatrix} t \\ 10 - t/4 \\ 15 \\ 5 - t/6 \end{bmatrix} = \begin{bmatrix} 24 \\ 4 \\ 15 \\ 1 \end{bmatrix}.$$

The third iteration begins.

Step 1. Solving the system $\mathbf{yB} = [1, 1, 1, 1]$, we obtain $\mathbf{y} = [\frac{7}{24}, \frac{1}{4}, \frac{5}{24}, \frac{1}{6}]$.

Step 2. Now we are looking for nonnegative integers a_1, a_2, a_3, a_4 such that

$$25.5a_1 + 22.5a_2 + 20a_3 + 15a_4 \leq 91$$

$$\frac{7}{24}a_1 + \frac{1}{4}a_2 + \frac{5}{24}a_3 + \frac{1}{6}a_4 > 1 .$$

The method to be discussed later establishes the nonexistence of such integers. We conclude that our current solution is optimal.

Clearly, this strategy is applicable to any cutting-stock problem with a single raw width. Its success hinges on an efficient subroutine for finding nonnegative integers $a_1, a_2, \ldots, a_m$ satisfying (13.3) or establishing their nonexistence.

We are about to present an algorithm that does a little more: it solves the problem:

$$\begin{aligned} &\text{maximize} && \sum_{i=1}^{m} y_i a_i \\ &\text{subject to} && \sum_{i=1}^{m} w_i a_i \leq r \\ &&& a_i = \text{nonnegative integer} \qquad (i = 1, 2, \ldots, m). \end{aligned} \tag{13.4}$$

This problem is known as the *knapsack problem.* (The name derives from a frivolous interpretation where a narrow knapsack of height r has to be filled with food packages, the ith brand having height w_i and value y_i.)

Solving the Knapsack Problem

In the usual notation, the knapsack problem is recorded as

$$\begin{aligned} &\text{maximize} && \sum_{i=1}^{m} c_i x_i \\ &\text{subject to} && \sum_{i=1}^{m} a_i x_i \le b \\ & && x_i = \text{nonnegative integer} \qquad (i = 1, 2, \ldots, m). \end{aligned} \tag{13.5}$$

Thus the variables are denoted by $x_1, x_2, \ldots, x_m$ rather than $a_1, a_2, \ldots, a_m$. Similarly, the coefficients w_i in (13.4) are replaced by a_i in (13.5); the coefficients y_i in (13.4) are replaced by c_i in (13.5); and the right-hand side is denoted by b rather than r. In typical applications, such as the cutting-stock problem, the numbers $a_1, a_2, \ldots, a_m$ and b are positive. Now we may assume that the numbers $c_1, c_2, \ldots, c_m$ are positive (variables x_i with $c_i \le 0$ could be deleted at once). In the frivolous interpretation, each a_i is thought of as the height of a food package, whereas c_i represents the corresponding value. Thus the ratio c_i/a_i amounts to the value per inch of the ith brand. In an intuitive sense, the desirability of including the ith brand in the knapsack increases with this ratio. We shall refer to c_i/a_i as the *efficiency* of the variable x_i. Without loss of generality, we may assume that the variables are subscripted in order of decreasing efficiency:

$$c_1/a_1 \ge c_2/a_2 \ge \cdots \ge c_m/a_m. \tag{13.6}$$

Finally, note that every optimal solution of (13.5) satisfies

$$b - \sum_{i=1}^{m} a_i x_i < a_m \tag{13.7}$$

for otherwise x_m could be increased by one unit. We shall refer to feasible solutions satisfying (13.7) as *sensible.*

The key to our method lies in a certain way of enumerating all the sensible solutions. We shall illustrate it on the problem

$$\begin{aligned} &\text{maximize} && 4x_1 + 5x_2 + 5x_3 + 2x_4 \\ &\text{subject to} && 33x_1 + 49x_2 + 51x_3 + 22x_4 \le 120 \\ & && x_1, x_2, x_3, x_4 = \text{nonnegative integer.} \end{aligned}$$

The reader may easily verify that this problem has precisely 13 sensible solutions; we shall describe them by the diagram in Figure 13.1. Diagrams of this kind are called *enumeration trees.* The small circles are the *nodes* of the tree; the leftmost node is the *root* and each of the 13 rightmost nodes is a *leaf*. (For typographical reasons, this tree grows in a direction seldom found in nature: from left to right.) Proceeding from the root to a leaf in four stages, we specify first a value of x_1, then a value of

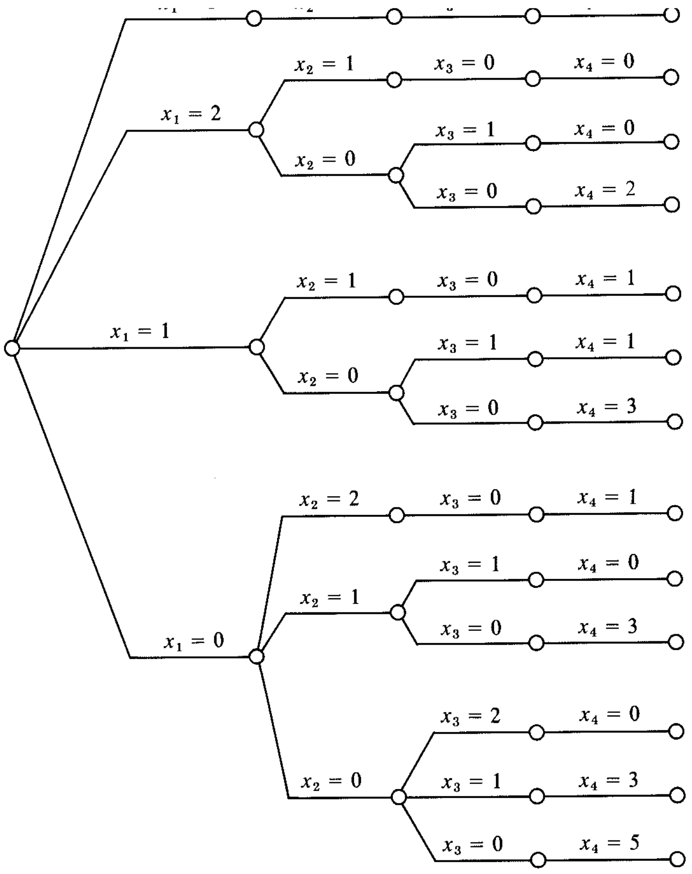

Figure 13.1

and vice versa. Note also that, whenever two or more branches grow from the same node, the higher branches correspond to higher values of x_j. In general, the solution corresponding to the *topmost* leaf is obtained by making x_1 as large as possible and, once x_1 has been fixed, making x_2 as large as possible, and so on. Formally, with $\lfloor z \rfloor$ denoting the integer obtained by rounding down z, this solution is defined by the recursion

$$x_j = \left\lfloor \left(b - \sum_{i=1}^{j-1} a_i x_i\right) \Big/ a_j \right\rfloor \qquad (j = 1, 2, \ldots, m).$$

In particular, $x_1 = \lfloor b/a_1 \rfloor$. We begin with this solution. If it weren't for certain shortcuts, we would scan through the list of leaves down to the bottom, keeping track of the best solution $x_1^*, x_2^*, \ldots, x_m^*$ found so far, and replacing it with a better solution whenever one comes up.

The scanning instructions make use of the tree structure. From each leaf that has just been examined, we backtrack toward the root step by step, keeping a constant lookout for as yet unexplored branches. When one is found, we reverse direction and proceed toward a new leaf; at each junction, we resolve possible ties by preferring higher branches to lower ones. As the reader may verify by consulting Figure 13.1, an algebraic description of this procedure is as follows. Having just examined a sensible solution $x_1, x_2, \ldots, x_m$, we set k equal to $m - 1$ and, if necessary, keep reducing k until $x_k > 0$ is found. Then we replace x_k by $x_k - 1$ and define the values of $x_{k+1}, x_{k+2}, \ldots, x_m$ recursively by

$$x_j = \left\lfloor \left(b - \sum_{i=1}^{j-1} a_i x_i\right) \Big/ a_j \right\rfloor.$$

The shortcuts will prevent us from moving along branches that are *obviously* hopeless. More precisely, suppose that the current best solution $x_1^*, x_2^*, \ldots, x_m^*$ yields

$$\sum_{i=1}^{m} c_i x_i^* = M$$

and that, backtracking from some $x_1, x_2, \ldots, x_m$ toward the root, we have just found the largest k such that $k \leq m - 1$ and $x_k > 0$. Proceeding as before, we would set

$$\begin{aligned} \bar{x}_i &= x_i \qquad \text{for all} \quad i = 1, 2, \ldots, k-1 \\ \bar{x}_k &= x_k - 1 \end{aligned}$$

and begin to examine various choices of $\bar{x}_{k+1}, \bar{x}_{k+2}, \ldots, \bar{x}_m$ leading to sensible solutions $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_m$. Before doing so, let us ask whether any of these solutions has a fighting chance against $x_1^*, x_2^*, \ldots, x_m^*$. By (13.6), each of the variables $x_{k+1}, x_{k+2}, \ldots, x_m$ has an efficiency of *at most* c_{k+1}/a_{k+1}, and so

$$\sum_{i=k+1}^{m} c_i \bar{x}_i \leq \frac{c_{k+1}}{a_{k+1}} \sum_{i=k+1}^{m} a_i \bar{x}_i.$$

Hence the assumption

$$\sum_{i=1}^{m} a_i \bar{x}_i \leq b$$

implies

$$\sum^{m} c_i \bar{x}_i \leq \sum^{k} c_i \bar{x}_i + \frac{c_{k+1}}{a}\left(b - \sum^{k} a_i \bar{x}_i\right). \tag{13.8}$$

has a chance of improving $x_1^*, x_2^*, \ldots, x_m^*$, and so we might as well forget about them. In terms of the enumeration tree, the inequality

$$\sum_{i=1}^{k} c_i\bar{x}_i + \frac{c_{k+1}}{a_{k+1}}\left(b - \sum_{i=1}^{k} a_i\bar{x}_i\right) \leq M \tag{13.9}$$

makes the path specified by $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k$ *not* worth exploring any further. In fact, if all the coefficients $c_1, c_2, \ldots, c_m$ are positive *integers*, then M is an integer and (13.9) may be replaced by the weaker inequality

$$\sum_{i=1}^{k} c_i\bar{x}_i + \frac{c_{k+1}}{a_{k+1}}\left(b - \sum_{i=1}^{k} a_i\bar{x}_i\right) < M + 1. \tag{13.10}$$

Let us see how useful these shortcuts turn out to be in our example. We initialize by

$$\begin{aligned} x_1 &= \lfloor 120/33 \rfloor = 3 \\ x_2 &= \lfloor (120 - 99)/49 \rfloor = 0 \\ x_3 &= \lfloor (120 - 99)/51 \rfloor = 0 \\ x_4 &= \lfloor (120 - 99)/22 \rfloor = 0 \end{aligned}$$

and file the initial solution as our current best:

$$x_1^* = 3, \quad x_2^* = x_3^* = x_4^* = 0 \quad \text{and} \quad M = 12. \tag{13.11}$$

Beginning with $k = 3$, we reduce k until $k = 1$ with $x_k > 0$ is found. Then we change $x_1 = 3$ into $x_1 = 2$. Before exploring the branch $x_1 = 2$ any further, let us test inequality (13.10) with $k = 1$ and $\bar{x}_1 = 2$. Since the left-hand side equals

$$8 + \frac{5}{49}(120 - 66) \doteq 13.5$$

which is *not* smaller than $M + 1 = 13$, the branch *may* be worth exploring. Computing

$$\begin{aligned} x_2 &= \lfloor (120 - 66)/49 \rfloor = 1 \\ x_3 &= \lfloor (120 - 115)/51 \rfloor = 0 \\ x_4 &= \lfloor (120 - 115)/22 \rfloor = 0 \end{aligned}$$

we indeed find an improvement over (13.11). Hence, we replace (13.11) by

$$x_1^* = 2, \quad x_2^* = 1, \quad x_3^* = x_4^* = 0 \quad \text{and} \quad M = 13. \tag{13.12}$$

Again, we begin with $k = 3$ and reduce k until $k = 2$ with $x_k > 0$ is found. Then we change $x_2 = 1$ into $x_2 = 0$. To find out whether the path $x_1 = 2, x_2 = 0$ is worth exploring further, we test inequality (13.10) with $k = 2$ and $\bar{x}_1 = 2, \bar{x}_2 = 0$. The left-hand side equals

$$8 + \frac{5}{51}(120 - 66) \doteq 13.3$$

which is smaller than $M + 1 = 14$. Hence, we reduce k further until the next k with $x_k > 0$. Having found $k = 1$, we replace $x_1 = 2$ by $x_1 = 1$. Is the branch $x_1 = 1$ worth exploring? Testing (13.10) with $k = 1$ and $\bar{x}_1 = 1$ shows that

$$4 + \frac{5}{49}(120 - 33) \doteq 12.9 < 14$$

and so the answer is negative. The branch $x_1 = 0$ would do even worse:

$$\frac{5}{49} \cdot 120 \doteq 12.2 < 14.$$

Hence, the search terminates and we conclude that (13.12) is an optimal solution.

In terms of the enumeration tree, not exploring the obviously hopeless branches may be viewed as pruning these branches off. In effect, the enumeration tree of Figure 13.1 reduces to the much smaller tree shown in Figure 13.2.

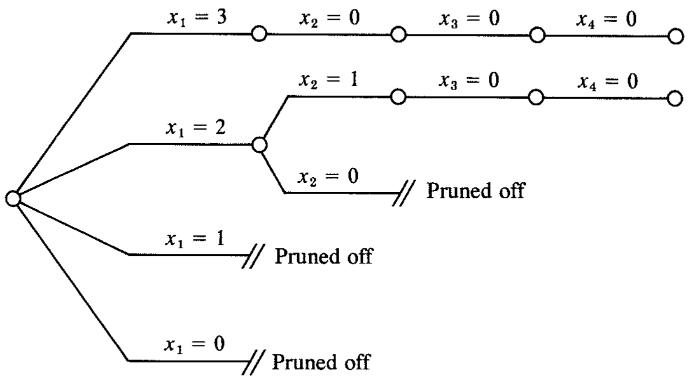

Figure 13.2

Note that having pruned off the branch $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k$, we can prune off $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k - 1$ at once, without performing the appropriate test. The point is that the relevant inequality,

$$\sum_{i=1}^{k-1} c_i\bar{x}_i + c_k(\bar{x}_k - 1) + \frac{c_{k+1}}{a_{k+1}}\left(b - \sum_{i=1}^{k-1} a_i\bar{x}_i - a_k(\bar{x}_k - 1)\right) < M + 1. \qquad (13.13)$$

is already implied by (13.10). Indeed, the left-hand side of (13.10) exceeds the left-hand side of (13.13) by which is nonnegative by virtue of (13.6). Of course, this observation brings about further computational savings.

This method, known as *branch-and-bound*, is described succintly in Box 13.1.

BOX 13.1 Branch-and-Bound Method for Solving the Knapsack Problem

Step 1. [Initialize.] Set $M = 0, k = 0$.

Step 2. [Find the most promising extension of the current branch.] For $j = k + 1, k + 2, \ldots, m$, set

$$x_j = \left\lfloor \left(b - \sum_{i=1}^{j-1} a_i x_i \right) \Big/ a_j \right\rfloor.$$

Then replace k by m.

Step 3. [An improved solution obtained?] If $\sum_{i=1}^{m} c_i x_i > M$, then replace M by $\sum_{i=1}^{m} c_i x_i$ and replace $x_1^*, x_2^*, \ldots, x_m^*$ by $x_1, x_2, \ldots, x_m$.

Step 4. [Backtrack to the next branch.]

a. If $k = 1$, then stop; otherwise replace k by $k - 1$.

b. If $x_k = 0$, then return to a; otherwise replace x_k by $x_k - 1$.

Step 5. [Branch worth exploring?] If (13.10) with x_i in place of $\bar{x}_i$ fails, then return to step 2; otherwise return to step 4.

Of course, if some of the coefficients c_i are not integers, then test (13.10) in step 5 must be replaced by (13.9). An alternative, and preferable, course of action is to find a positive d such that each dc_i is an integer, and then replace $c_1, c_2, \ldots, c_m$ by $dc_1, dc_2, \ldots, dc_m$. For instance, the objective function in the knapsack problem

$$\begin{array}{ll} \text{maximize} & \frac{1}{5}x_1 + \frac{1}{7}x_2 + \frac{1}{9}x_3 + \frac{1}{9}x_4 \\ \text{subject to} & 17.25x_1 + 12.75x_2 + 10x_3 + 15x_4 \le 90 \\ & \qquad x_1, x_2, x_3, x_4 = \text{nonnegative integer} \end{array}$$

may be replaced by $63x_1 + 45x_2 + 35x_3 + 35x_4$.

An alternative method is based on a simple idea known as the principle of *dynamic programming*. Dynamic programming algorithms, however, tend to get quite tedious as the right-hand side b increases. In experiments with the cutting-stock problem, P. C. Gilmore and R. E. Gomory (1963) found the branch-and-bound method about five times as fast as dynamic programming. On the examples discussed here, dynamic programming would certainly require far more work than we would care to go through. We leave this line of attack for problems at the end of this chapter (13.4–13.8); for more information, the reader is referred to Chapter 6 of R. S. Garfinkel and G. L. Nemhauser (1972).

Finding a Good Initial Solution

In solving the cutting-stock problem by the revised simplex method with delayed column generation, it is desirable to begin with a near-optimal basic feasible solution: if the initial value of the objective function is close to the optimum value, then the number of simplex iterations required to reach the optimum is likely to be small. We are about to describe a very fast procedure for finding reasonably good initial solutions. This procedure is based on the assumption that it pays to get the wide finals out of the way first, letting the versatile narrow ones compete for the leftovers.

Again, we shall denote the single raw width by r and say that the order summary calls for b_i finals of width w_i $(i = 1, 2, \ldots, m)$. Furthermore, we shall assume that the final widths are arranged in decreasing order:

$$w_1 > w_2 > \cdots > w_m.$$

Our procedure constructs the initial $\mathbf{B}$ and $\mathbf{x}$ in m iterations. At the beginning of iteration j, we have a certain set R containing precisely $m - j + 1$ of the subscripts $1, 2, \ldots, m$; for each subscript i in R, we have a nonnegative number b_i', interpreted as the residual demand for finals of width w_i. (To initialize, we set $b_i' = b_i$ for all i.) In the jth iteration, we define the jth column $\mathbf{a} = [a_1, a_2, \ldots, a_m]^T$ of $\mathbf{B}$ recursively by

$$a_i = \begin{cases} 0 & \text{if } i \notin R \\ \left\lfloor \left(r - \sum_{k=1}^{i-1} w_k a_k \right) \Big/ w_i \right\rfloor & \text{if } i \in R. \end{cases}$$

Our initial solution x will use this pattern until some residual demand b_k' gets fully satisfied. To put it differently, the component x_j corresponding to the jth column of $\mathbf{B}$ will equal the smallest of the ratios b_i'/a_i with $i \in R$ and $a_i > 0$. Thus, $x_j a_i \le b_i'$ for all $i \in R$ and $x_j a_k = b_k'$ for at least one $k \in R$. We delete this subscript k from R, replace each remaining b_i' by $b_i' - x_j a_i$ and proceed to the $(j + 1)$th iteration.

For illustration, we return to our first cutting-stock example with $r = 100$, $w_1 = 45$, $w_2 = 36$, $w_3 = 31$, $w_4 = 14$, $b_1 = 97$, $b_2 = 610$, $b_3 = 395$, $b_4 = 211$. Here the initialization procedure works as follows.

Iteration 1. We find $a_1 = \lfloor 100/45 \rfloor = 2$, $a_2 = \lfloor 10/36 \rfloor = 0$, $a_3 = \lfloor 10/31 \rfloor = 0$, $a_4 = \lfloor 10/14 \rfloor = 0$. Now $x_1 = \frac{97}{2} = 48.5$. We delete the subscript 1 from R and obtain $b_2' = 610$, $b_3' = 395$, $b_4' = 211$.

Iteration 2. We find $a_1 = 0$, $a_2 = \lfloor 100/36 \rfloor = 2$, $a_3 = \lfloor 28/31 \rfloor = 0$, $a_4 = \lfloor 28/14 \rfloor = 2$. Now $x_2 = \min(\frac{610}{2}, \frac{211}{2}) = 105.5$. We delete the subscript 4 from R and obtain $b_2' = 399$, $b_3' = 395$.

Iteration 4. We find $a_1 = 0$, $a_2 = 0$, $a_3 = \lfloor 100/31 \rfloor = 3$, $a_4 = 0$. Now x_4 3

Thus we have produced the basic feasible solution with

$$\mathbf{B} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 2 & 0 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{x}_B^* = \begin{bmatrix} 48.5 \\ 105.5 \\ 199.5 \\ 131.67 \end{bmatrix}.$$

Note that this solution is only one simplex iteration away from the optimum.

This procedure is closely related to a method known as *first-fit decreasing*, which produces reasonably good integer-valued solutions for any bin-packing problem. In the jth iteration of first-fit decreasing, we find a way of cutting the jth raw. The iteration begins with certain residual demands $b'_1, b'_2, \ldots, b'_m$; we specify the cutting pattern recursively by

$$a_i = \min \begin{cases} b'_i \\ \left\lfloor \left(r - \sum_{k=1}^{i-1} w_k a_k \right) \Big/ w_i \right\rfloor \end{cases}$$

for $i = 1, 2, \ldots, m$, then replace each b'_i by $b'_i - a_i$, and proceed to the $(j + 1)$th iteration. For instance, first-fit decreasing applied to our example yields the following solution:

48 times	$a_1 = 2$,	$a_2 = 0$,	$a_3 = 0$,	$a_4 = 0$
once	1	1	0	1
105 times	0	2	0	2
199 times	0	2	0	0
once	0	1	2	0
131 times	0	0	3	0.

A remarkable guarantee of performance for first-fit decreasing has been established by D. S. Johnson (1973): if the integer-valued optimal solution uses n raws, then the integer-valued solution found by first-fit decreasing uses at most $\frac{11}{9}n + 4$ raws. (The proof is very difficult. Nevertheless, it is easy to show that the bound cannot be improved except possibly for the dangling 4; see problem 13.3.) It follows that our initialization procedure always gets within 23% of the fractional optimum, give or take a few raws. □

Additional Complications

The problems arising in the paper industry may be more complicated than the simple examples discussed here. The complications stem from three major sources.

(i) The raws may come in several different widths $r_1, r_2, \ldots, r_N$. Producing a raw of width r_k costs c_k dollars; the problem is to fill the summary of orders at the least cost.

(ii) Raws of different widths are cut on different machines. Limited machine availability may allow no more than v_k raws of width r_k to be cut.

(iii) The number of cutting knives on each machine is limited, usually to six or so. Consequently, a cutting pattern specified by $a_1, a_2, \ldots, a_m$ is admissible only if $\sum a_i$ does not exceed a predetermined bound.

All these complications can be handled by easy modifications of the method just described, and, therefore, we shall comment on them only briefly. First, let us consider the linear programming problem

$$\text{minimize} \quad \mathbf{cx} \qquad \text{subject to} \quad \mathbf{Ax} = \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}$$

arising from a cutting-stock problem with N raw widths. The columns of $\mathbf{A}$ come in N different groups; columns in group k specify cutting patterns with $\sum_{i=1}^{m} w_i a_i \leq r_k$ and the corresponding components of $\mathbf{c}$ equal c_k. To generate the entering column, we have to find nonnegative integers $a_1, a_2, \ldots, a_m$ and a subscript k such that $\sum_{i=1}^{m} w_i a_i \leq r_k$ and $\sum_{i=1}^{m} y_i a_i > c_k$. This task can be accomplished by solving N knapsack problems, the kth of which reads

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{m} y_i a_i \\ \text{subject to} & \sum_{i=1}^{m} w_i a_i \leq r_k \\ & a_i = \text{nonnegative integer} \end{array} \tag{13.14}$$

and then comparing the optimum values $z_1^*, z_2^*, \ldots, z_N^*$ against $c_1, c_2, \ldots, c_N$. The optimal solution with the largest $z_k^* - c_k$ yields the desired entering column. (In fact, the N enumeration trees can be combined into one. We leave the details for problem 13.9.) Secondly, if at most v_k raws of width w_k can be cut ($k = 1,2, \ldots, N$), then we enlarge our set of constraints by N additional inequalities: the kth of them reads $\sum x_j \leq v_k$, with the summation running through all the variables x_j in the kth group. Now each basic solution includes $m + N$ rather than m basic variables; in step 2 of each simplex iteration, we compare z_k^* against $c_k - y_{m+k}$ rather than c_k. (The product $\mathbf{ya}$ equals $z_k^* + y_{m+k}$ in this case.) Finally, if the number of cutting knives is limited, then the enumeration trees can be easily modified to explore only the admissible patterns.

EXPERIMENTAL RESULTS

The second paper by P. C. Gilmore and R. E. Gomory (1963) on the cutting-stock problem contains a wealth of experimental findings. The results estimate the expected difficulty of a typical selecting from a class of paper industry problems. raw rolls of only one size, around 200 in. or less. The number of different final widths ranged from 20 to 40. These widths, generally between 20 in. and 80 in., were specified to $\frac{1}{4}$ in. The number of cutting knives was limited to five, seven, or nine.

The *average* number of simplex iterations for these problems came to about 130. However, the *individual* iteration counts showed a large variance over the sample, from 20 to more than 300. Such variance is quite common: linear programming problems looking quite alike often behave very differently from each other when subjected to the simplex method. As might be expected, problems with smaller numbers of final widths tended to require fewer iterations. Nevertheless, this trend was rather erratic; for example, the average number of iterations required by the 35-final problems was 197, whereas the corresponding figure for the 40-final problems was only 161. (No procedure for finding good initial basic solutions was used; the simplex method seems to find a fairly good fractional solution in the first few iterations.)

Some of the questions answered by Gilmore and Gomory concerned the criterion for selecting a new cutting pattern. For instance, one might either accept the first encountered pattern with $\sum y_i a_i > 1$ or insist on the pattern maximizing $\sum y_i a_i$. Which of the two methods is better? It seems plausible that the first method leads to a larger number of iterations, but it is also clear that the time per iteration is smaller for this method. Without experimental findings, the outcome of the trade-off between the total number of iterations and the time per individual iteration might have been difficult to predict. The Gilmore–Gomory results show that finding the true maximum in the knapsack problem is well worth the effort. In all but one of the 20 problems, the *overall* time required by the second method was smaller. In fact, the average amount of time per problem required by the second method was about half the corresponding figure for the first method.

In terms of Chapter 4, the second method is nothing but the largest-coefficient pivoting rule, whereas the first method is a relative of the smallest-subscript rule. Thus it is only natural to speculate about the largest-improvement rule in the cutting-stock context. A straightforward implementation of this rule would require enumerating an enormous number of different cutting patterns in each iteration, in which case the means would defeat the purpose. However, there is a natural way of mimicking the largest-improvement rule and, at the same time, actually *reducing* the computational effort per iteration. The idea is simple. Some of the final widths are ordered in very small quantities. If our new cutting pattern yields any of these low-demand widths, then the pattern is bound to be used only a few times. Consequently, the resulting improvement won't be very impressive. Led by this observation, Gilmore and Gomory suggested the use of the *median method*. With this method, the final widths are divided into two equally large groups: the low-demand group and the high-demand group. Then, at every second iteration, the cutting pattern is chosen that (i) yields only the high-demand finals and (ii) maximizes the improvement rate among all such patterns. (Note that the corresponding knapsack problem, involving only half of the original variables, is easier to solve.) The median method turned out to be faster in 13 of the 20 basic test problems; furthermore, the problems on which the performance of the median method was worse than that of the original method were mostly the problems that required a small amount of time anyway. The average running time per problem was cut down to about 40% of the original figures.

In each particular instance of the cutting-stock problem, a certain percentage of the paper consumed in the form of raw rolls is directed to the finals, and the rest goes to waste. In the test problems, the waste percentage seems to have been unpredictable, ranging from as little as 0.1% to as much as 10%. An interesting observation is that the high-waste problems were generally solved faster than the low-waste problems. In a typical low-waste problem, the waste dropped sharply in the beginning and then proceeded to creep slowly towards the optimum: although