3.

r	s	1	
3/4	1/4	−3/2	$= -x$
7	1	−10	$= -y$
−14	−2	12	$= u$

4.

s	y	1	
0	2	−6	$= -z$
1	−12	−9	$= -r$
3/2	−21/2	−9/2	$= -x$
−2	−10	54	$= u$

5.

r	s	1	
4/3	7/4	−9	$= -x$
−18/5	7/4	−10/9	$= -y$
−5/4	−3/5	5/3	$= u$

6.

r	z	s	1	
1/2	−3/4	5/3	−2/5	$= -x$
−2/3	1/3	14/5	−1	$= -y$
−1/4	−1/2	−5/3	11/6	$= u$

10.4
BRANCH AND BOUND PROCEDURE

A method used in practice very often is the branch and bound method. This appears to be the most effective method currently known for general integer programming problems, but even this becomes unwieldy as the number of variables becomes large (say, more than a few hundred). Thus, the quest for an efficient method for solving integer programs remains. Although we only discuss the method for pure integer programs, the method carries over with simple modification to mixed integer programs.

Here is the idea: Suppose we have a pure integer program and solve the LP relaxation of the program, and suppose that the solution of the LP relaxation is $x = 3.6$, $y = 2$. This clearly cannot be the solution of the pure integer program, since the optimal value of x is not integral. Since we require that x be integral and there are no integers between 3 and 4, the optimal value of x must either be less than or equal to 3 or greater than or equal to 4. So we set up two different linear programs and solve them. One of them is the LP relaxation together with the constraint $x \leq 3$, which we call LP1 for the present, and the other will be the LP relaxation of the original program together with the constraint $x \geq 4$, which we call LP2 for the present. The formation of LP1 and LP2 is called branching. We are assured that our optimal solution must be in one of the constraint sets determined by these two modified linear programs by the previous discussion. Furthermore, *all* feasible solutions are in one or the other of these constraint sets. Thus, nothing is lost by forming these two linear subprograms and solving them.

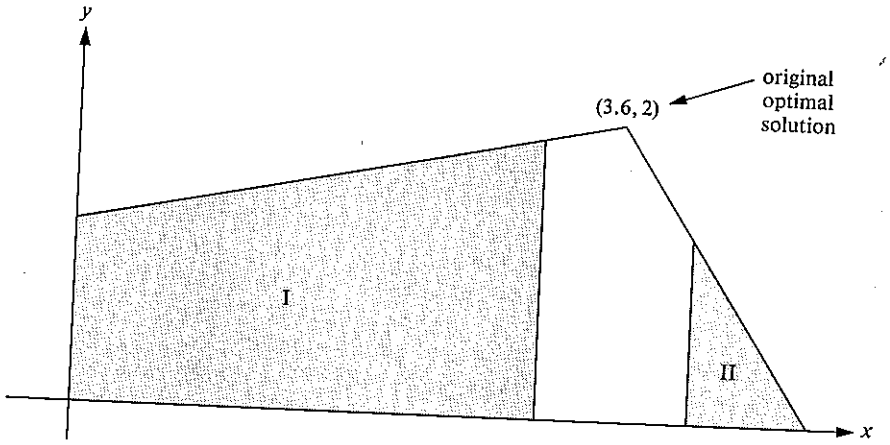

Figure 10.4.

Pictorially, all this is shown in Fig. 10.4. The constraint set of LP1 is on the left, while the constraint set of LP2 is on the right. There are no integer solutions of the original program in the unshaded portion of the picture. We have formed new linear programs by adding cuts, only these cuts are parallel to the axes. We now solve these linear programs and see if we get integral solutions to either of them. If we get an integral solution to one program but not the other, then we put the integral solution aside. Suppose the value of the objective function, u, at this point is a; then, the optimal value of u must be at least as big as a. Thus, a serves as a lower bound for the optimal solution.

We now work on the other program, splitting it up into subprograms by adding cuts parallel to the y axis if the optimal solution of this program has x nonintegral, or by adding cuts parallel to the x axis if the optimal solution has y nonintegral. We keep splitting the subprograms into further subprograms by adding cuts, and we try to generate better and better *integer* solutions. When we can no longer do this, we are done. Our best integer solution is our optimal solution.

At first glance, this seems pretty unwieldy, because as we keep splitting the subprograms, we have more and more linear programs to deal with. However, there are shortcuts that make it unnecessary to analyze all our subprograms. We need only take a suitable sample of them and work from there. The details of this will be given as we illustrate this method in the next few examples. In all cases, whenever we solve a linear subprogram formed and obtain a variable, say v, that is fractional, we form two new linear programs from this one. The first is obtained by adding the constraint $v \leq b$, where b is the largest whole number less than or equal to v. The

second is obtained by adding the constraint $v \geq b + 1$. We need not do this for *all* the variables that have fractional values, just for one each time. We are guaranteed that every time we do this we will lose no integral solutions of the linear subprogram. Furthermore, if we join the integral solutions of the subprograms together, we get the integral solutions of the original linear program. Let us illustrate.

EXAMPLE 1 Suppose we want to

$$\begin{aligned} \text{Maximize} \quad & u = 3x + 4y, \\ \text{s.t.} \quad & 4x + 3y \leq 13, \\ & 3x + 2y \leq 7, \\ & x, y \geq 0 \quad \text{and integral.} \end{aligned}$$

If we solve the LP relaxation,

$$\begin{aligned} \text{Maximize} \quad & u = 3x + 4y, \\ \text{s.t.} \quad & 4x + 3y \leq 13, \\ & 3x + 2y \leq 7, \\ & x, y \geq 0. \end{aligned} \tag{27}$$

we obtain $x = 0$ and $y = 3.5$. The objective at this point is 14. In view of the fact that y must be integral, it follows that either $y \leq 3$ or $y \geq 4$. We form two subprograms: the first subprogram, LP1, consists of (27) together with the constraint $y \leq 3$; the second linear program, LP2, consists of the (27) together with the constraint $y \geq 4$. We indicate this as in Fig. 10.5. The circles are called nodes, and lines joining nodes to other nodes are called

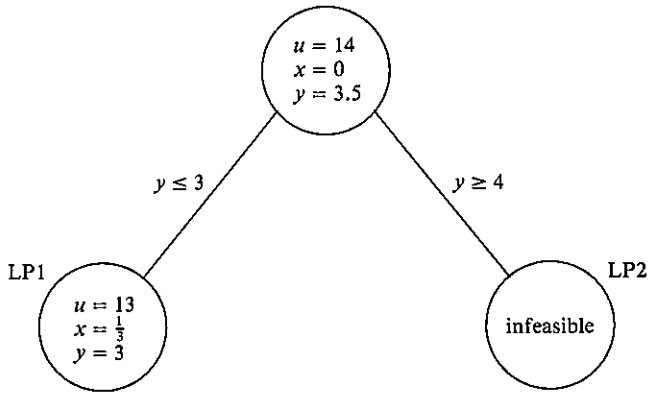

Figure 10.5.

branches. If we solve LP1, we find that the optimal solution is $x = 1/3$ and $y = 3$. Here, the objective function is 13. If we solve LP2, we find that the program is infeasible. Since we can get no further information from the node corresponding to LP2, we drop it from further consideration. Any node dropped in this way, or any node no longer in use, is called a *fathomed* node. Any other node is called a *dangling* node. The node corresponding to LP1 is dangling at this point.

Since the optimal solution of LP1 requires that $x = 1/3$ and we know that x must be integral, we branch on LP1 to form two programs, LP3 and LP4. LP3 consists of LP2 together with the constraint $x \leq 0$, and LP4 consists of LP2 together with the constraint $x \geq 1$. We note that since all variables are ≥ 0, the constraint $x \leq 0$ in LP3 forces x to be equal to zero. Our picture now is shown in Fig. 10.6.

Solving LP3, we find that $x = 0$, $y = 3$, and $u = 12$. We have found an integral solution that makes the objective function equal to 12. Thus, at this point our best integral solution is for LP3, and we know that the optimal value of the original linear program must at least 12. What about LP4? Is it possible that there is an integral solution to LP4 that is greater than 12? Theoretically, there is nothing to stop this from happening, and so we must solve LP4 also. There we find the solution $x = 1$ and $y = 2$. The objective value at this point is 11. The question facing us now is whether we branch

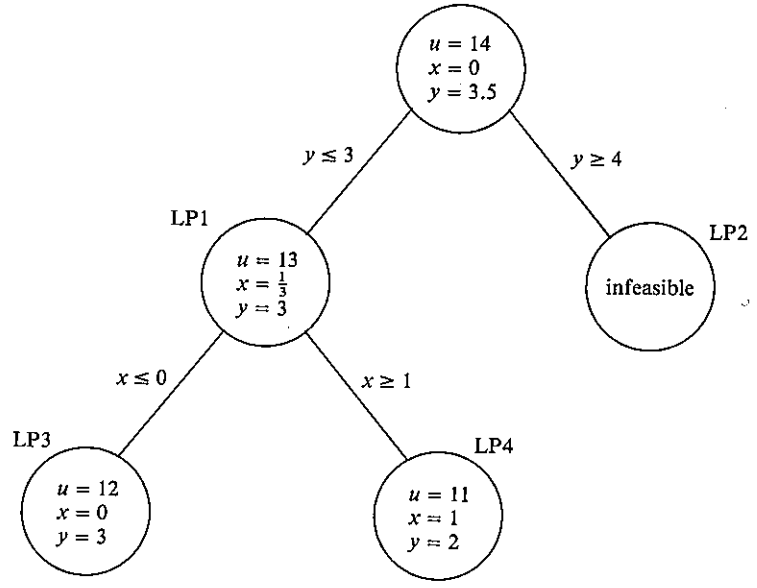

Figure 10.6.

again on these nodes to perhaps get better solutions. The answer is no, for the following reason: Whenever we add a constraint to a maximum program, the value of the objective function can only stay the same or decrease. Thus, to branch on node 4 makes no sense since our new program can only have an objective ≤ 11, and we already have a better value of the objective at LP3. Thus, node 4 is fathomed, as we can get no further useful information from it. Since adding a constraint can only decrease the objective in LP3, we have also fathomed that node. Thus, it no longer pays to branch further on any nodes, and the current best integral solution, $x = 0$, $y = 3$, is the optimal solution.

Let us give another example.

EXAMPLE 2 We wish to

$$\begin{aligned} \text{Maximize} \quad & u = 4x + 5y + 6z, \\ \text{s.t.} \quad & 3x + 2y + z \leq 9, \\ & 2x + y + 4z \leq 7, \\ & x, y, z \geq 0 \text{ and integral.} \end{aligned}$$

The solution process is summarized in Fig. 10.7. Let us go through the process. When we solve the LP relaxation, we obtain $u = 25$, $x = 0$, $y \approx 4.14$, $z \approx 0.71$. Since y is not integral, we may branch on y. The two branches are obtained by adding the constraint $y \leq 4$ to the LP relaxation to get node 1, and adding the constraint $y \geq 5$ to the LP relaxation to get node 2. Solving the program at node 1, we get $u = 24.6$, $x = 0.1$, $y = 4$, $z = 0.7$. The program corresponding to node 2 is infeasible. Now, node 1 is dangling, and we may branch on x to get nodes 3 and 4. Solving the program corresponding to node 3, we get $u = 24.5$, $x = 0$, $y = 4$, $z = 0.75$. Solving the program corresponding to node 4, we obtain $u = 21$, $x = 1$, $y \approx 2.7$, $z \approx 0.57$. So far we have no integral solutions. Nodes 3 and 4 are still dangling. We branch on node 3 to get nodes 5 and 6. The solution of the program at node 5 is integral. There $u = 20$, $x = z = 0$, and $y = 4$. The solution at node 6 is also integral: $u = 21$, $x = 0$, $y = 3$, $z = 1$. At this point, our best integral solution occurs at node 6. Nodes 5 and 6 are fathomed. Branching further on either of them will only serve to decrease the objective. Node 4 is still dangling; but there is no sense in branching on that node, since branching can only lead to an objective ≤ 21, and we have already obtained an integral solution where $u = 21$. Thus, we may consider node 4 fathomed. Since all nodes are fathomed, we have reached our optimal solution. It occurs at node 6, and it is $u = 21$ when $x = 0$, $y = 3$, and $z = 1$.

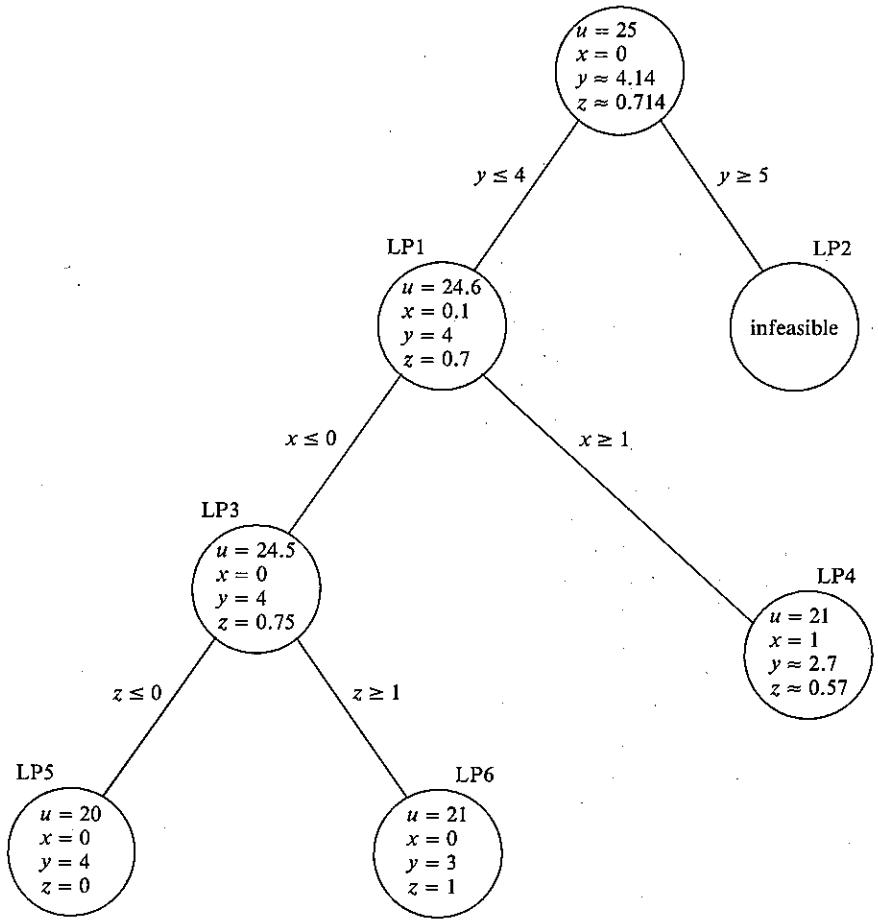

Figure 10.7.

One advantage of the branch and bound procedure is that as we proceed with it we often generate integral solutions along the way that are pretty good. If the solutions are acceptable to us, even though they are not optimal, we may stop at that point. This is especially useful when the branch and bound tree becomes very large. Also, sometimes we can obtain an integral solution of the program by inspection. This helps because then many nodes that we might have had to fathom otherwise will not need to be fathomed, because we know that they will not benefit us. For example, if in some problem we obtained $u = 45$ when $x = y = z = 3$, and we did this by inspection, then any node where u is less than or equal to 45 need not be studied. It is considered fathomed.

But, as we said earlier, however good the method is, it can lead to very big trees even for simple problems. Let us illustrate this by solving an example from earlier in the chapter. That example,

$$\begin{aligned} \text{Maximize} \quad & u = 58x + 200y, \\ \text{s.t.} \quad & 12x + 40y \le 87, \\ & x, y \ge 0 \quad \text{and integral}, \end{aligned}$$

when solved by the branch and bound procedure has the solution tree given in Fig. 10.8. Notice that our optimal integral solution does not occur until the 11th node. And this problem has only one constraint! The Gomory method, which is usually more time consuming, solves this problem very quickly. This should illustrate why it is useful to have many methods to draw when solving integer programs.

There are quite a few other methods that one can talk about, but since this chapter was meant only to give an overview of the types of methods used, we will not go into any others. We should, however, make a few comments. When we solve the LP relaxation of a pure integer program, we may, if we wish round the solution. This might be desirable for several reasons: (1) The problem may be large, and solving the problem by the branch and bound method may use up large amounts of computer time, which might not be financially justifiable or even available; (2) Rounding may give a feasible solution that is close enough to optimality to be usable; and (3) Rounding is fast. So, if a solution is needed right way, this might be the way to go. The question that naturally comes up is, just how good is the rounded solution, *assuming* it is feasible? To answer this, let us call the optimal objective value of the LP relaxation u^*, the optimal objective value of the pure integer program u_I^*, and the objective value at the rounded solution u_R. Clearly,

$$u_R \le u_I^* \le u^*. \tag{28}$$

Suppose we compute the quantity $(u^* - u_R)/u_R$. Call this value d. Thus, $du_R = u^* - u_R$. If we subtract u_R from each term in (28), then we have

$$0 \le u_I^* - u_R \le u^* - u_R,$$

which may be rewritten as

$$0 \le u_I^* - u_R \le du_R.$$

Adding u_R to the inequality yields

$$u_R \le u_I^* \le (1 + d)u_R.$$

This statement, which tells us that the optimal solution of the original integral program is between the value of the objective function at the rounded solution and $(1 + d)$ times this value, gives us a measure of how

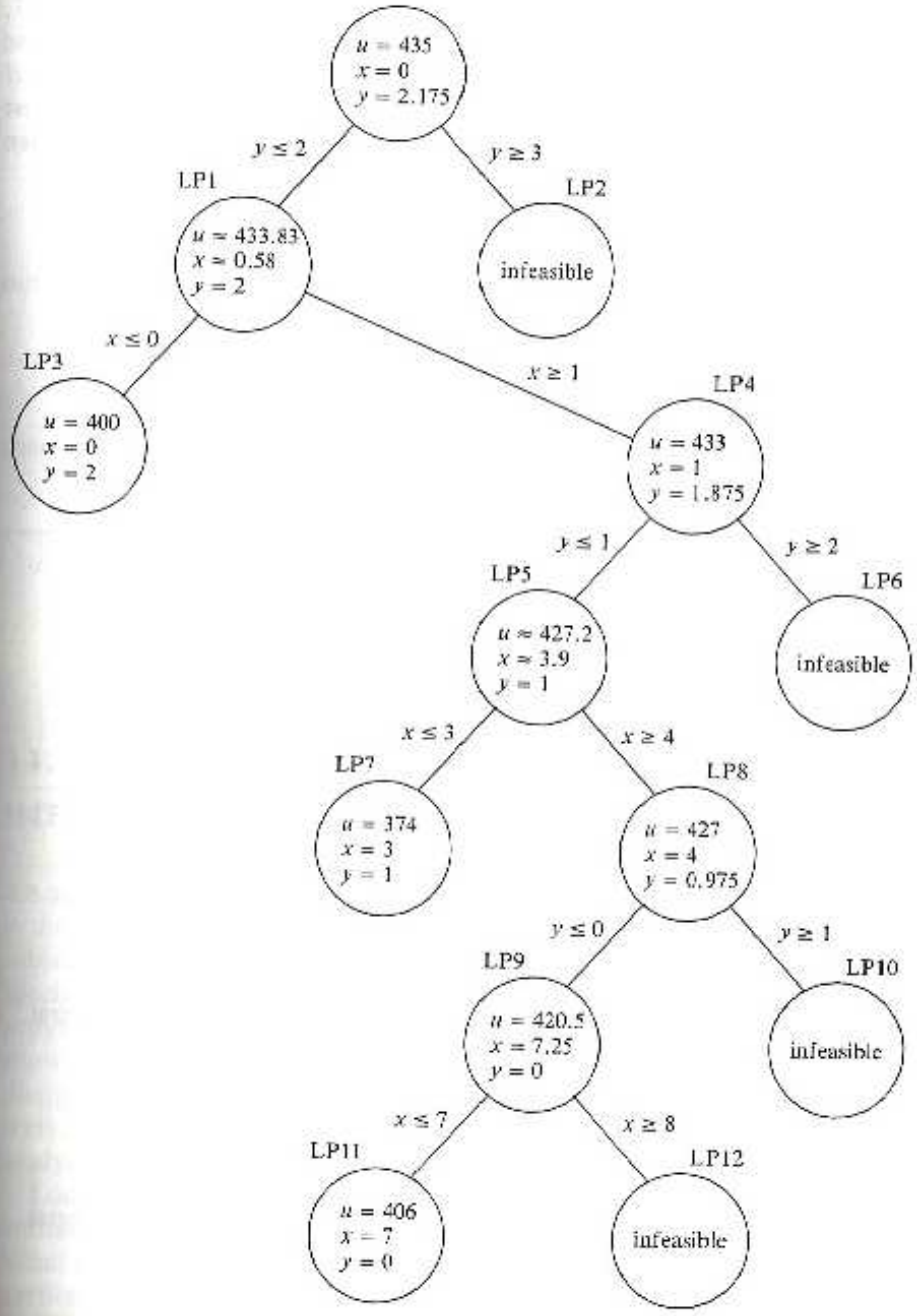

Figure 10.8.

close the rounded solution is to the true optimal solution. Thus, if $d = 0.01$, our true solution is within 1% of the rounded solution, and so we are close to the optimal solution. For that reason, it probably pays to round when d is sufficiently small. In a similar manner, if u_I represents the current best integral value of u when using the branch and bound procedure, then when

$$\frac{u^* - u_I}{u_I}$$

is small, say β, we can be assured that the optimal integral solution is within $100\beta\%$ of u_I. That is, $u_I \le u_I^* \le (1 + \beta)u_I$.

EXERCISES 10.4

1. Use the branch and bound method to solve each of the following. Draw the branch and bound tree for each problem.

(a) Minimize $u = 5x + y$, s.t.
$3x + 2y \ge 4$,
$x \ge 2$,
$y \ge 0$,
x and y are integral.

(b) Minimize $u = 5x + y$, s.t.
$1.5 \le x \le 3.4$,
$2.1 \le y \le 2.7$,
x and y are integral.

(c) Maximize $u = 2x - y$, s.t.
$x + 2y \le 5$,
$3x - y \le 7$,
$x, y \ge 0$ and integral.

(d) Maximize $u = 3x + 4y$, s.t.
$2x + 3y \le 7$,
$x, y \ge 0$ and integral.

(e) Maximize $u = 2x + 3y$, s.t.
$x \ge y$,
$x + 2y \le 6$,
$2x + y \le 8$,
$x, y \ge 0$ and integral.

(f) Maximize $u = 5x - y - z$, s.t.
$2x + y + z \le 4$,
$x + 3y + 4z \le 1$,
$x, y, z \ge 0$ and integral.

(g) Maximize $u = 3x + 4y + 5z$, s.t.
$2x + 3y + z \le 6$,
$x + 3y + 4z \le 5$,
$x, y, z \ge 0$ and integral.

(h) Maximize $u = 3x - 2y + z$, s.t.
$4x + y + z \le 6$,
$3x + 2y + 3z \le 4$,
$x, y, z \ge 0$ and integral.

CHAPTER 11
NETWORK ANALYSIS

11.1
INTRODUCTION AND DEFINITIONS

An area of mathematics that has grown tremendously in the last 100 years is the subject of graph theory. The number of varied applications of this subject is enormous and continues to grow. In this chapter, we will study a special subdivision of graph theory that is closely connected to linear programming—network analysis. We will be brief, since our goal is only to show how linear programming may be used in other areas. More detailed discussions of the topics in this chapter may be found in operations research texts, management science texts, and, of course, graph theory and network analysis texts.

Loosely speaking, a *graph* is a collection of objects, called nodes or vertices (represented by dots or circles), together with a set of edges. What characterizes an edge is that it joins two vertices. (But not every two vertices need be joined by an edge.) Several examples are given in Fig. 11.1. In Fig. 11.1a, the graph has four vertices, labelled 1, 2, 3, and 4, and two edges. In Fig. 11.1b, the graph has four vertices and three edges; while in Fig. 11.1c, the graph has three vertices and one edge.

(Blank page)