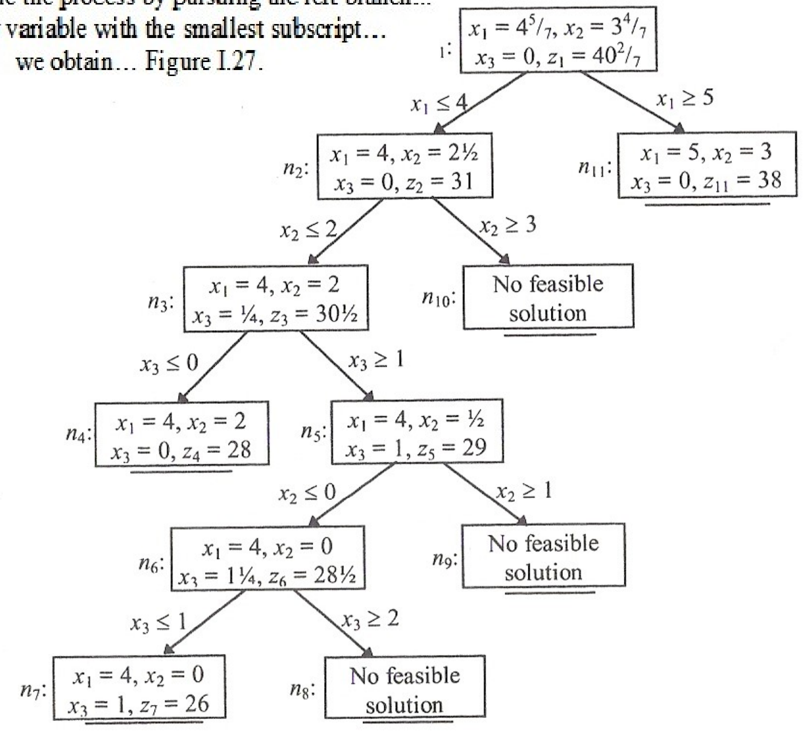
modern branch and bound techniques, no more than two branches emanate from each of the nodes, as distinct from the original approach of Land and Doig (1960).

...if we continue the process by pursuing the left branch...
noninteger variable with the smallest subscript...
we obtain... Figure I.27.



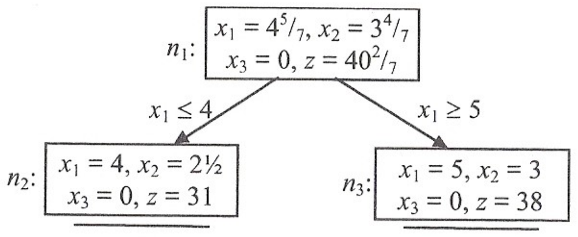
1: $x_1 = 4^5/_7, x_2 = 3^4/_7$, $x_3 = 0, z_1 = 40^2/_7$
$x_1 \le 4$
$x_1 \ge 5$
n_2: $x_1 = 4, x_2 = 2½$, $x_3 = 0, z_2 = 31$
n_{11}: $x_1 = 5, x_2 = 3$, $x_3 = 0, z_{11} = 38$
$x_2 \le 2$
$x_2 \ge 3$
n_3: $x_1 = 4, x_2 = 2$, $x_3 = ¼, z_3 = 30½$
n_{10}: No feasible solution
$x_3 \le 0$
$x_3 \ge 1$
n_4: $x_1 = 4, x_2 = 2$, $x_3 = 0, z_4 = 28$
n_5: $x_1 = 4, x_2 = ½$, $x_3 = 1, z_5 = 29$
$x_2 \le 0$
$x_2 \ge 1$
n_6: $x_1 = 4, x_2 = 0$, $x_3 = 1¼, z_6 = 28½$
n_9: No feasible solution
$x_3 \le 1$
$x_3 \ge 2$
n_7: $x_1 = 4, x_2 = 0$, $x_3 = 1, z_7 = 26$
n_8: No feasible solution

Figure I.27

6.2 Search strategies

Since each node of the solution tree represents a linear programming problem that must be solved in the process, it is obviously important to keep the size of the tree as small as possible. As demonstrated in the previous section, this is accomplished by stopping at a node whose associated problem has an integral solution, is infeasible, or is fathomed (value dominated). A good branch and bound method will bring about such favorable situations as early as possible in the search. In example 2 in the previous section we pursued each path until integrality or infeasibility was reached. This is called a *depth-first search* strategy. The *depth* (or *level*) of a node is the length of the path leading to it from the top node; in our example, node n_6 is at depth 4 whereas node n_{10} is at depth 2. Similarly, the solution tree has grown to a depth of 5. In general, a depth first strategy will pursue each node, using some branching strategy, until integrality, infeasibility, or

fathoming has occurred, and then backtrack until the first node is encountered, which can be further pursued. This strategy will bring us to high levels (deep in the tree) early in the search. By contrast, a *breadth-first search* strategy explores all nodes at one level before proceeding to the next. If in our example a breadth first search strategy had been used with the same branching strategy (branch on the noninteger variable with the smallest index and pursue the left branch first), we would have obtained the branch and bound tree displayed in Figure I.28.



n_1: $x_1 = 4^5/_7, x_2 = 3^4/_7$, $x_3 = 0, z = 40^2/_7$
$x_1 \le 4$
$x_1 \ge 5$
n_2: $x_1 = 4, x_2 = 2½$, $x_3 = 0, z = 31$
n_3: $x_1 = 5, x_2 = 3$, $x_3 = 0, z = 38$

Figure I.28

I.28?

In Figure I.41, node n_2 is fathomed due to the z-value of 38 obtained at node n_3. We conclude that in this particular example, the breadth-first search strategy was much better than the depth-first search strategy in the sense that the tree is much smaller. In practice, however, it appears that depth-first strategies are more advantageous than breadth-first. One reason for this is that the number of nodes grows exponentially as the level increases, and since feasible (integer) solutions are typically found deeper into the tree, depth-first search tends to find integer solutions sooner than breadth-first search. Another appealing feature of depth-first search is related to the way in which the linear programming problems at each node are solved. Assuming that the simplex method is used for solving the problems, an immediate successor of a node has the same set of constraints as its predecessor, plus the one additional constraint generated in the branching. The dual simplex method is therefore ideally suited for this situation and we will now illustrate how the solution at node n_2 in our example is obtained once the solution at the top node n_1 has been found. The linear programming relaxation of the problem in Example 2 is

$x_1, x_2, x_3 \in N_0$

$$\begin{aligned} P_{LP}:\ & \text{Max } z = z_1 = 4x_1 + 6x_2 + 10x_3 \\ \text{s.t.}\ & 2x_1 + x_2 + 2x_3 \le 13 \\ & 3x_1 - 2x_2 - 4x_3 \ge 7 \\ & x_1,\ x_2,\ x_3 \ge 0 \end{aligned}$$

$\mathbf{T}_3$:

x_1	x_2	x_3	S_1	E_2	S_1^*	S_2^*	S_3^*	1
0	1	0	0	0	0	1	0	2
1	0	0	0	0	1	0	0	4
0	0	0	1	0	−2	−1	−2	3
0	0	1	0	0	0	0	1	0
0	0	0	0	1	3	−2	−4	1
0	0	0	0	0	4	6	10	28

leau $\mathbf{T}_3$ corresponds to the solution at node n_4, which is all-integer so that no
her branching is necessary. To illustrate the branching leading to a node with
easible solution, consider again the tableau $\mathbf{T}_1$ and pursue the branch $x_2 \geq 3$
ling to node n_{10} of Figure I.40. Expressed in terms of nonbasic variables, $x_2 \geq 3$
written as $2\frac{1}{2} - 2x_3 - \frac{1}{2}E_2 - {}^3/_2\, S_1^* \geq 3$ or $2x_3 + \frac{1}{2}E_2 + {}^3/_2\, S_1^* \leq -\frac{1}{2}$. Adding a
k variable S_2^{**} to the left-hand side of this constraint, the tableau is

$\mathbf{T}_1''$:

x_1	x_2	x_3	S_1	E_2	S_1^*	S_2^{**}	1
0	1	2	0	½	³/₂	0	2½
1	0	0	0	0	1	0	4
0	0	0	1	−½	−⁷/₂	0	2½
0	0	2	0	½	³/₂	1	−½
0	0	2	0	3	13	0	31

fourth row in $\mathbf{T}_1''$ with a negative right-hand side of −½ is the only eligible
t row. However, all coefficients on the left-hand side in this row are
negative, so that no dual simplex pivot step is possible. We conclude that no
ible solution exists for node n_{10}.

eneral, with the optimal simplex tableau available for a node, the solutions for
mmediate successor nodes are obtained by first adding a row to the tableau that
esponds to the *branching cut* (the additional constraint), and then employing
dual simplex method to determine an optimal solution for the next node. In our
nple, only one dual simplex step was needed for each successor node, and in
eral only a few iterations are required to find the optimal solution of the next
e. It is worth noting that although developing a branch and bound tree involves
ing a linear programming problem at each node of the tree, each problem
ept for the top node can start the optimization from the optimal solution of the
nediate predecessor node, whose solution was made infeasible by the addition

of the new constraint (a so-called "hot start"). We observe that every branching cut will be binding immediately after it has been applied, if the next solution is feasible. The reason is that if the branching cut were not binding, it could be deleted, resulting in the same solution as previously, which contradicts the fact that both branching cuts cut off the previous solution. However, further down the tree, branching cuts may become again non-binding. As an example, consider the node n_3 in Figure I.40. The constraint $x_3 \geq 1$ on the right branch out of n_3 is no longer binding at node n_6, where $x_3 = 1\frac{1}{4}$. In general, suppose that at a node we branch on the same variable that has already been branched on at some predecessor node. The new branching constraint must then be tighter than the previous one on the same variable, which could therefore be deleted. In the dual simplex tableau, the new row would therefore simply replace the row that corresponds to the previous branching on the same variable. Since branching cuts are $\leq$ or $\geq$ constraints, it follows that the number of additional rows in the tableau never exceeds twice the number of integer restricted variables in the problem.

Back to the construction of a branch and bound tree, there are two decisions to be made each time a branching is to be done, *viz*.:

- which node to pursue next (*node selection strategy*), and
- which variable to branch on (*branch selection strategy*).

We will discuss these two types of strategies in the next two sections.

6.2.1 Node Selection

A number of strategies exist for selecting the next node to branch from. For simplicity of the exposition, assume again that the original problem has a maximization objective. If so far in the search one or more nodes have been encountered for which the solution is integer, we choose the one with the highest objective function value (ties are broken arbitrarily) and call it the *incumbent node* or solution. The collection of nodes that have no branches leading out of them forms the set S. Initially, only the top node is in S.

The nodes in the set S fall into four categories.

(1) The solution at a node $n_i \in S$ is integer, in which case we will not branch further from this node. If the solution is better than that of the existing incumbent(s) n_k, then n_i replaces n_k; if it is as good as n_k it becomes another incumbent.

(2) The node $n_i \in S$ represents a problem that has no feasible solution, making branching on it pointless, since any successor problem will also lack feasible

(3) solutions.

(4) The node n_i is fathomed, i.e., its noninteger solution is no better than the incumbent solution. Again, its potential successors have poorer solutions than the incumbent and are therefore not of interest.

(5) The node n_i has a noninteger solution with an objective function value that is better than that of the incumbent, so that branching from this node might lead to an integer optimal solution. Such a node is called *live* or *active*.

In summary, out of the four possible node types—integer, infeasible, fathomed, and live—we only branch from live nodes. Defining $L \subset S$ as the set of live nodes, the node selection problem addresses the choice of a node $n_i \in L$.

We have already described two node selection strategies: the breadth-first strategy where all live nodes at a given level are considered before nodes on lower levels are examined, and the depth-first strategy where the next node to be considered is a live successor of the latest node that was explored. Since backtracking is needed if the node that was explored last is not live, the strategy is better described as "depth-first with backtracking"; or *last in, first out* (LIFO), borrowing a concept from inventory management. Although a depth-first strategy can be expected to perform better than a breadth-first strategy on average, both of these strategies would be outperformed, again on average, by strategies that make better use of the information gathered during the construction of the tree. One such strategy is the *best-bound-first* strategy that selects the live node with the largest z-value. Formally, the best-bound-first strategy selects the node n_k such that $z_k = \max\{z_\ell \colon n_\ell \in L\}$.

Another node selection strategy involves the estimate $\hat{z}_\ell$ of the integer optimal solution corresponding to node n_ℓ. Such an estimate can be computed based on the expected *degradation* expressing the deterioration of z_ℓ by requiring the solution point at node n_ℓ to be integral. More specifically, let the solution at node n_i include $\tilde{x}_j = \lfloor \tilde{x}_j \rfloor + f_j$ with $f_j \neq 0$. Using some user-selected coefficients p_j^- and p_j^+, we estimate the decrease in the objective function of $p_j^- f_j$ for branching left at node n_i and of $p_j^+(1 - f_j)$ if branching right. The coefficients p_j^- and p_j^+ can either be user-specified or estimated, e.g., by using dual information at node n_i or information from previous branchings on x_j. A *best-estimate* search strategy selects the node n_k such that $\hat{z}_k = \max\{\hat{z}_\ell : n_\ell \in L\}$. Denoting by $\underline{z}_{\text{IP}}$ the objective function value of the incumbent solution, the *quick-improvement* strategy attempts

to quickly improve on the incumbent solution by selecting the node n_k such that $k = \arg\max\{(z_\ell - \underline{z}_{\text{IP}}) \,/\, (z_\ell - \hat{z}_\ell)\}$. For further discussion and details on node selection, the reader is referred to Nemhauser and Wolsey (1988).

6.2.2 Branch Selection

Once a node $n_k \in L$ has been selected for further exploration, the next decision concerns the choice of variable to be branched on. Clearly, this variable, while required to be integer, must currently have a noninteger value. In the example above we used the simple strategy of branching on the noninteger variable with lowest index, an obviously arbitrary rule. Another strategy is to branch on the noninteger variable x_k with the "most fractional" value. Formally, let $\tilde{\mathbf{x}} = [\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n]$ be the solution at the chosen live node, and let $\tilde{x}_j = \lfloor \tilde{x}_j \rfloor + f_j$, so that f_j denotes the *fractional* part of $\tilde{x}_j$. The *most fractional* strategy would branch on the variable whose present value is farthest from the nearest integer or, equivalently, has the value closest to ½, i.e., a variable x_k with $k = \underset{j=1,\ldots,n}{\arg\min}\,\{|f_j - ½|\}$. Unfortunately, experience has failed to identify robust methods for branch selection, and in practice user-specified priorities are employed. Other methods involve degradation measures similar to those in Subsection 6.2.1. More involved methods that employ penalties and use more elaborate computations regarding the penalty coefficients p_j^- and p_j^+ have not turned out to improve the overall efficiency of the search if the additional computational effort it takes to apply them is taken into account.

6.3 A General Branch and Bound Procedure

We are now able to formulate a general branch and bound algorithm. The problem to be solved is an all-integer or mixed-integer programming problem P_{IP} with a maximization objective; its linear programming relaxation is called P_{LP}. It is assumed that specific node and branch selection strategies have been chosen. The algorithm is initialized with node n_1 that includes the optimal linear programming relaxation $\bar{\mathbf{x}}_{\text{LP}}$ with objective value $\bar{z}_{\text{LP}}$. Given that $\bar{\mathbf{x}}_{\text{LP}}$ does not satisfy all of the integrality conditions (otherwise $\bar{\mathbf{x}}_{\text{LP}} = \bar{\mathbf{x}}_{\text{IP}}$ is optimal for the (mixed) integer programming problem as well), set $S := L := \{n_1\}$, $t := 1$, and $\underline{z} := -\infty$.

A General Branch and Bound Algorithm

Step 1: Is $L = \emptyset$? If yes: If $\underline{z} > -\infty$, then the solution $\bar{\mathbf{x}}_{\text{IP}} = \underline{\mathbf{x}}$ with objective value $\bar{z}_{\text{LP}} = \underline{z}$ is optimal, otherwise the integer programming problem has no feasible solution.

If no: Go to Step 2.

Step 2: Choose some node $n_i \in L$ (according to some prescribed criterion) with solution $\mathbf{x}^i$ and objective value z_i.

Step 3: Branch on some variable x_j (according to some prescribed criterion) to the nodes n_{t+1} (with $x_j \le \lfloor x_j \rfloor$) and n_{t+2} (with $x_j \ge \lceil x_j \rceil$). set $L := L \setminus \{n_i\}$, and $S := S \cup \{n_{t+1}, n_{t+2}\} \setminus \{n_i\}$. Solve the linear programming problems at n_{t+1} and n_{t+2}. The results are solutions $\mathbf{x}^{t+1}$ and $\mathbf{x}^{t+2}$ with objective values z_{t+1} and z_{t+2}, respectively.

Step 4: Is $\mathbf{x}^{t+1}$ feasible? If yes: Go to Step 5.

If no: Go to Step 7.

Step 5: Does $\mathbf{x}^{t+1}$ satisfy the integrality requirements of the original problem?

If yes: Go to Step 6.

If no: Set $L := L \cup \{n_{t+1}\}$ and go to Step 7.

Step 6: Is $z_{t+1} > \underline{z}$? If yes: Set $\underline{\mathbf{x}} := \mathbf{x}^{t+1}$, $\underline{z} := z_{t+1}$, and go to Step 7.

If no: Go to Step 7.

Step 7: Repeat Steps 4-6 with $t + 2$ instead of $t + 1$; then set $t := t + 2$ and go to Step 1.

The key to the algorithm is the updating procedure of the sets S and L. In each step when the procedure branches from some node n_i to two nodes n_{t+1} and n_{t+2}, the set of end nodes S is updated to include the new nodes n_{t+1} and n_{t+2} and to exclude n_i. From the set of live nodes L, the node from which the branching takes place is deleted in Step 3, and node n_{t+1} (or n_{t+2}) is added to the set in Step 5 only if its solution is feasible (Step 4) but not yet integer (Step 5).

Example: As an illustration of the above algorithm, consider again the problem of Example 2 of Section 6.1. Here, we choose the node with the best objective value to be branched on next, and we select the "most fractional" variable for the branching. The resulting branch and bound tree is shown in Figure I.29.

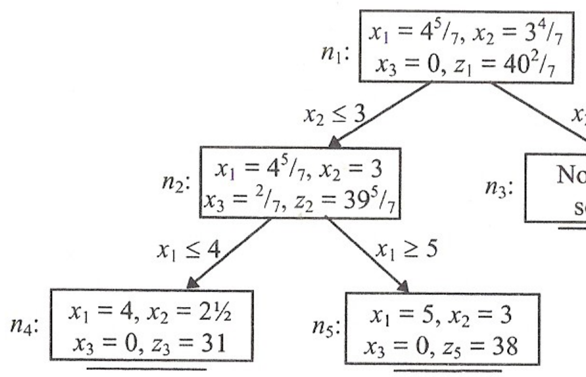

n_1: $x_1 = 4^5/_7$, $x_2 = 3^4/_7$
$x_3 = 0$, $z_1 = 40^2/_7$
$x_2 \le 3$
$x_2 \ge 4$
n_2: $x_1 = 4^5/_7$, $x_2 = 3$
$x_3 = {}^2/_7$, $z_2 = 39^5/_7$
n_3: No feasible solution
$x_1 \le 4$
$x_1 \ge 5$
n_4: $x_1 = 4$, $x_2 = 2½$
$x_3 = 0$, $z_3 = 31$
n_5: $x_1 = 5$, $x_2 = 3$
$x_3 = 0$, $z_5 = 38$

Figure I.29

No further branching is possible due to infeasibility at node n_3, fathoming for node n_4 and integrality for node n_5. The set of L of live nodes during the algorithm is $\{n_1\}$, $\{n_2\}$, $\{n_4\}$, and $\emptyset$. The solution tree contains only five nodes, whereas eleven nodes were required for the same problem with a depth-first node strategy. If a best-bound-first node and a lowest-index variable strategy had been selected, only three nodes would have been required, obtaining the same tree as in Figure I.28. This may demonstrate the difficulty in finding the best strategies to use when a particular problem is being solved.

There is an interesting analogy between cutting plane algorithms and branch and bound methods. We may view the branching constraints as vertical or horizontal cutting planes designed to cut off areas of the feasible region that do not contain any integer points. It is also possible to mix in regular cutting planes with the branching at the nodes of a branch and bound tree. This approach is called *branch and cut*, and is typically used for zero-one problems as well as problems with special structures. The idea is to find valid inequalities for the original problem, which are violated at some nodes in a branch and bound tree. These valid inequalities are then added at these nodes in a cutting plane fashion, thus generating new nodes from which branching can be done as usual. This could be accomplished by introducing an additional step between the existing Steps 2 and 3 in the general branch and bound algorithm above. For details, the reader is referred to Hoffman and Padberg (1985) and Rardin (1998).

6.4 Difficult Problems

When addressing the issue of computational complexity, it is well known that