7

Branch and Bound

7.1 DIVIDE AND CONQUER

Consider the problem:

$$z = \max\{cx : x \in S\}.$$

How can we break the problem into a series of smaller problems that are easier, solve the smaller problems, and then put the information together again to solve the original problem?

Proposition 7.1 *Let $S = S_1 \cup \ldots \cup S_K$ be a decomposition of S into smaller sets, and let $z^k = \max\{cx : x \in S_k\}$ for $k = 1, \ldots, K$. Then $z = \max_k z^k$.*

A typical way to represent such a divide and conquer approach is via an enumeration tree. For instance, if $S \subseteq \{0,1\}^3$, we might construct the enumeration tree shown in Figure 7.1.

Here we first divide S into $S_0 = \{x \in S : x_1 = 0\}$ and $S_1 = \{x \in S : x_1 = 1\}$, then $S_{00} = \{x \in S_0 : x_2 = 0\} = \{x \in S : x_1 = x_2 = 0\}$, $S_{01} = \{x \in S_0 : x_2 = 1\}$, and so on. Note that a leaf of the tree $S_{i_1 i_2 i_3}$ is nonempty if and only if $x = (i_1, i_2, i_3)$ is in S. Thus the leaves of the tree correspond precisely to the points of B^3 that one would examine if one carried out complete enumeration. Note that by convention the tree is drawn upside down with its root at the top.

Another example is the enumeration of all the tours of the traveling salesman problem. First we divide S the set of all tours on 4 cities into $S_{(12)}, S_{(13)}, S_{(14)}$ where $S_{(ij)}$ is the set of all tours containing arc (ij). Then $S_{(12)}$ is divided again into $S_{(12)(23)}$ and $S_{(12)(24)}$, and so on. Note that at the first level we have arbitrarily chosen to branch on the arcs leaving node 1, and at the

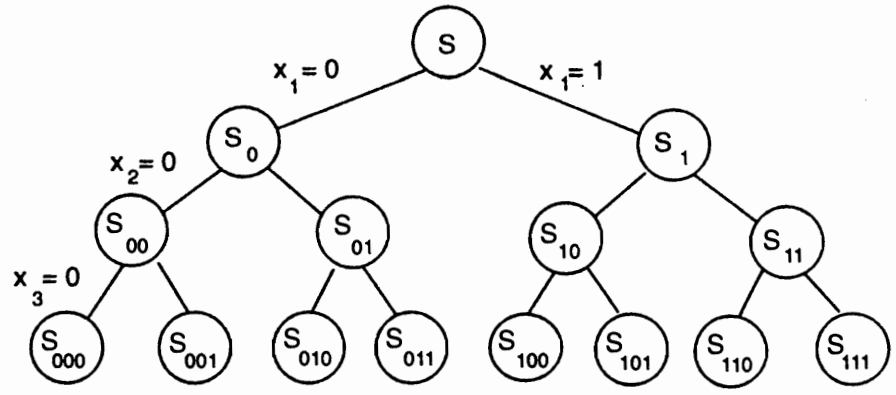

Fig. 7.1 Binary enumeration tree

second level on the arcs leaving node 2 that do not immediately create a subtour with the previous branching arc. The resulting tree is shown in Figure 7.2. Here the six leaves of the tree correspond to the $(n-1)!$ tours shown, where $i_1 i_2 i_3 i_4$ means that the cities are visited in the order i_1, i_2, i_3, i_4, i_1 respectively. Note that this is an example of multiway as opposed to binary branching, where a set can be divided into more than two parts.

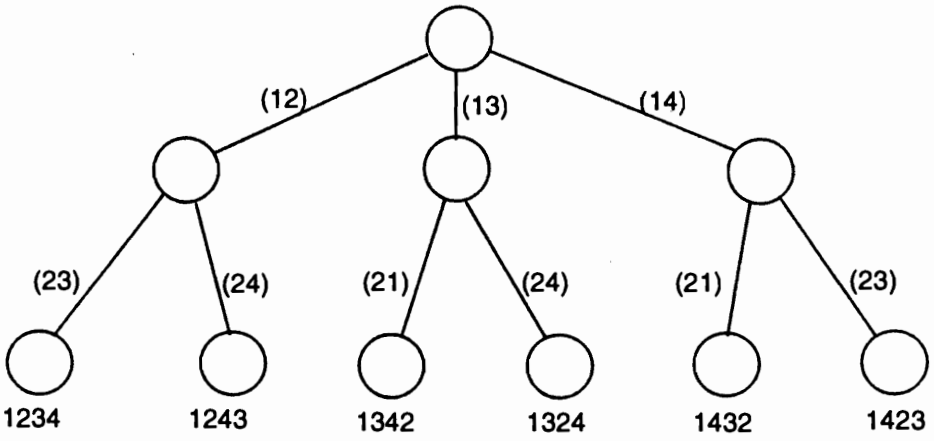

Fig. 7.2 *TSP* enumeration tree

7.2 IMPLICIT ENUMERATION

We saw in Chapter 1 that complete enumeration is totally impossible for most problems as soon as the number of variables in an integer program, or nodes in a graph exceeds 20 or 30. So we need to do more than just divide indefinitely. How can we use some bounds on the values of $\{z^k\}$ intelligently? First, how can we put together bound information?

Proposition 7.2 *Let $S = S_1 \cup \ldots \cup S_K$ be a decomposition of S into smaller sets, and let $z^k = \max\{cx : x \in S_k\}$ for $k = 1, \ldots, K$, $\overline{z}^k$ be an upper bound on z^k and $\underline{z}^k$ be a lower bound on z^k. Then $\overline{z} = \max_k \overline{z}^k$ is an upper bound on z and $\underline{z} = \max_k \underline{z}^k$ is a lower bound on z.*

Now we examine three hypothetical examples to see how bound information, or partial information about a subproblem can be put to use. What can be deduced about lower and upper bounds on the optimal value z and which sets need further examination in order to find the optimal value?

Example 7.1 In Figure 7.3 we show a decomposition of S into two sets S_1 and S_2 as well as upper and lower bounds on the corresponding problems.

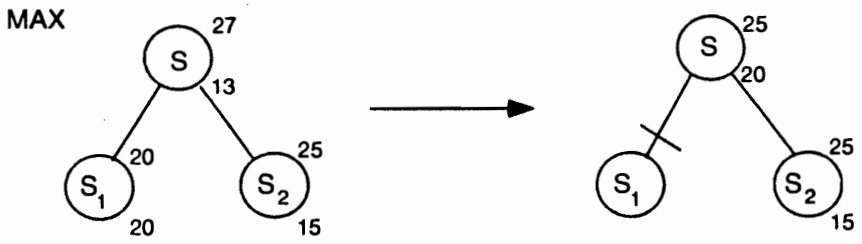

Fig. 7.3 Pruned by optimality

We note first that $\overline{z} = \max_k \overline{z}^k = \max\{20, 25\} = 25$ and $\underline{z} = \max_k \underline{z}^k = \max\{20, 15\} = 20$.

Second, we observe that as the lower and upper bounds on z_1 are equal, $z_1 = 20$, and there is no further reason to examine the set S_1. Therefore the branch S_1 of the enumeration tree can be pruned *by optimality.* ∎

Example 7.2 In Figure 7.4 we again decompose S into two sets S_1 and S_2 and show upper and lower bounds on the corresponding problems.

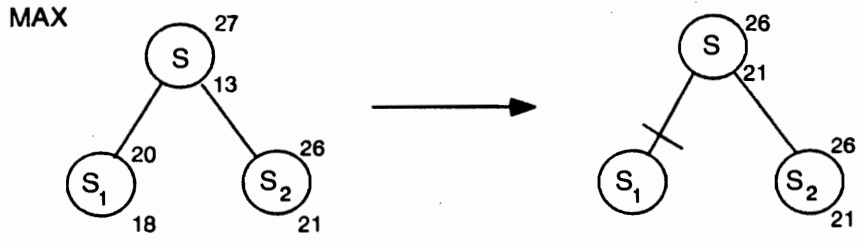

Fig. 7.4 Pruned by bound

We note first that $\overline{z} = \max_k \overline{z}^k = \max\{20, 26\} = 26$ and $\underline{z} = \max_k \underline{z}^k = \max\{18, 21\} = 21$.

Second, we observe that as the optimal value has value at least 21, and the upper bound $\overline{z}^1 = 20$, no optimal solution can lie in the set S_1. Therefore the branch S_1 of the enumeration tree can be pruned *by bound.* ∎

Example 7.3 In Figure 7.5 we again decompose S into two sets S_1 and S_2 with different upper and lower bounds.

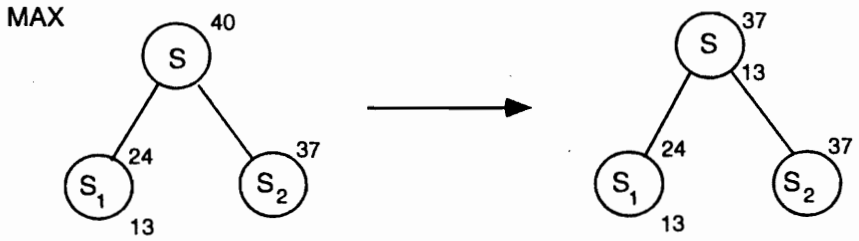

Fig. 7.5 No pruning possible

We note first that $\overline{z} = \max_k \overline{z}^k = \max\{24, 37\} = 37$ and $\underline{z} = \max_k \underline{z}^k = \max\{13, -\} = 13$. Here no other conclusion can be drawn and we need to explore both sets S_1 and S_2 further. ∎

Based on these examples, we can list at least three reasons that allow us to prune the tree and thus enumerate a large number of solutions implicitly.

(i) Pruning by optimality: $z_t = \{\max cx : x \in S_t\}$ has been solved.

(ii) Pruning by bound: $\overline{z_t} \leq \underline{z}$.

(iii) Pruning by infeasiblity: $S_t = \phi$.

If we now ask how the bounds are to be obtained, the reply is no different from in Chapter 2. The primal (lower) bounds are provided by feasible solutions and the dual (upper) bounds by relaxation or duality.

Building an implicit enumeration algorithm based on the above ideas is now in principle a fairly straightforward task. There are, however, many questions that must be addressed before such an algorithm is well-defined. Some of the most important questions are:

What relaxation or dual problem should be used to provide upper bounds? How should one choose between a fairly weak bound that can be calculated very rapidly and a stronger bound whose calculation takes a considerable time?

How should the feasible region be separated into smaller regions $S = S_1 \cup \ldots \cup S_K$? Should one separate into two or more parts? Should one use a fixed a priori rule for dividing up the set, or should the divisions evolve as a function of the bounds and solutions obtained en route?

In what order should the subproblems be examined? Typically there is a list of active problems that have not yet been pruned. Should the next one be chosen on a the basis of last-in first-out, of best/largest upper bound first, or of some totally different criterion?

These and other questions will be discussed further once we have seen an example.

7.3 BRANCH AND BOUND: AN EXAMPLE

The most common way to solve integer programs is to use implicit enumeration, or *branch and bound*, in which linear programming relaxations provide the bounds. We first demonstrate the approach by an example:

$$z = \max 4x_1 - x_2 \tag{7.1}$$
$$7x_1 - 2x_2 \leq 14 \tag{7.2}$$
$$x_2 \leq 3 \tag{7.3}$$
$$2x_1 - 2x_2 \leq 3 \tag{7.4}$$
$$x \in Z_+^2. \tag{7.5}$$

Bounding. To obtain a first upper bound, we add slack variables x_3, x_4, x_5 and solve the linear programming relaxation in which the integrality constraints are dropped. The resulting optimal basis representation is:

$$\begin{array}{llllll} \overline{z} = \max \frac{59}{7} & & -\frac{4}{7}x_3 & -\frac{1}{7}x_4 & & \\ x_1 & & +\frac{1}{7}x_3 & +\frac{2}{7}x_4 & = & \frac{20}{7} \\ & x_2 & & +x_4 & = & 3 \\ & & -\frac{2}{7}x_3 & +\frac{10}{7}x_4 & +x_5 = & \frac{23}{7} \\ x_1, & x_2, & x_3, & x_4, & x_5 \geq & 0. \end{array}$$

Thus we obtain an upper bound $\overline{z} = \frac{59}{7}$, and a nonintegral solution $(\overline{x}_1, \overline{x}_2) = (\frac{20}{7}, 3)$. Is there any straightforward way to find a feasible solution? Apparently not. By convention, as no feasible solution is yet available, we take as lower bound $\underline{z} = -\infty$.

Branching. Now because $\underline{z} < \overline{z}$, we need to divide or branch. How should we split up the feasible region? One simple idea is to choose an integer variable that is basic and fractional in the linear programming solution, and split the problem into two about this fractional value. If $x_j = \overline{x}_j \notin Z^1$, one can take:

$$S_1 = S \cap \{x : x_j \leq \lfloor \overline{x}_j \rfloor\}$$
$$S_2 = S \cap \{x : x_j \geq \lceil \overline{x}_j \rceil\}.$$

It is clear that $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \phi$. Another reason for this choice is that the solution $\bar{x}$ of $LP(S)$ is not feasible in either $LP(S_1)$ or $LP(S_2)$. This implies that if there is no degeneracy (i.e., multiple optimal LP solutions), then $\max\{\overline{z}_1, \overline{z}_2\} < \overline{z}$, so the upper bound will strictly decrease.

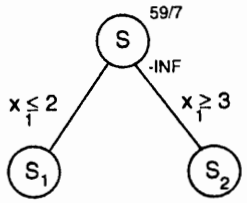

Fig. 7.6 Partial branch-and-bound tree 1

Following this idea, as $\bar{x}_1 = 20/7 \notin Z^1$, we take $S_1 = S \cap \{x : x_1 \leq 2\}$ and $S_2 = S \cap \{x : x_1 \geq 3\}$. We now have the tree shown in Figure 7.6. The subproblems (nodes) that must still be examined are called *active*.

Choosing a Node. The list of active problems (nodes) to be examined now contains S_1, S_2. We arbitrarily choose S_1.

Reoptimizing. How should we solve the new modified linear programs $LP(S_i)$ for $i = 1, 2$ without starting again from scratch?

As we have just added one single upper or lower bound constraint to the linear program, our previous optimal basis remains dual feasible, and it is therefore natural to reoptimize from this basis using the dual simplex algorithm. Typically, only a few pivots will be needed to find the new optimal linear programming solution.

Applying this to the linear program $LP(S_1)$, we can write the new constraint $x_1 \leq 2$ as $x_1 + s = 2, s \geq 0$, which can be rewritten in terms of the nonbasic variables as

$$-\tfrac{1}{7}x_3 - \tfrac{2}{7}x_4 + s = -\tfrac{6}{7}.$$

Thus we have the dual feasible representation:

$$\begin{array}{llllllll}
\overline{z}_1 = \max \tfrac{59}{7} & & -\tfrac{4}{7}x_3 & -\tfrac{1}{7}x_4 & & & & \\
x_1 & & +\tfrac{1}{7}x_3 & +\tfrac{2}{7}x_4 & & & = & \tfrac{20}{7} \\
& x_2 & & +x_4 & & & = & 3 \\
& & -\tfrac{2}{7}x_3 & +\tfrac{10}{7}x_4 & +x_5 & & = & \tfrac{23}{7} \\
& & -\tfrac{1}{7}x_3 & -\tfrac{2}{7}x_4 & & +s & = & -\tfrac{6}{7} \\
x_1, & x_2, & x_3, & x_4, & x_5, & s & \geq & 0.
\end{array}$$

After two simplex pivots, the linear program is reoptimized, giving:

$$\begin{array}{llllllll}
\overline{z}_1 = \max \tfrac{15}{2} & & & & -\tfrac{1}{2}x_5 & -3s & & \\
x_1 & & & & & +s & = 2 & \\
& x_2 & & & -\tfrac{1}{2}x_5 & +s & = \tfrac{1}{2} & \\
& & x_3 & & -x_5 & -5s & = 1 & \\
& & & x_4 & +\tfrac{1}{2}x_5 & +6s & = \tfrac{5}{2} & \\
x_1, & x_2, & x_3, & x_4, & x_5, & s & \geq 0 &
\end{array}$$

with $\overline{z}_1 = \frac{15}{2}$, and $(\overline{x}_1^1, \overline{x}_2^1) = (2, \frac{1}{2})$.

Branching. S_1 cannot be pruned, so using the same branching rule as before, we create two new nodes $S_{11} = S_1 \cap \{x : x_2 \leq 0\}$ and $S_{12} = S_1 \cap \{x : x_2 \geq 1\}$, and add them to the node list. The tree is now as shown in Figure 7.7.

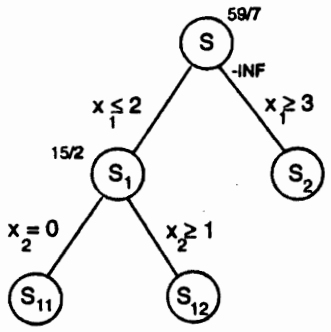

Fig. 7.7 Partial branch-and-bound tree 2

Choosing a Node. The active node list now contains S_2, S_{11}, S_{12}. Arbitrarily choosing S_2, we remove it from the node list and examine it in more detail.

Reoptimizing. To solve $LP(S_2)$, we use the dual simplex algorithm in the same way as above. The constraint $x_1 \geq 3$ is first written as $x_1 - t = 3, t \geq 0$, which expressed in terms of the nonbasic variables becomes:

$$\tfrac{1}{7}x_3 + \tfrac{2}{7}x_4 + t = -\tfrac{1}{7}.$$

From inspection of this constraint, we see that the resulting linear program

$$\begin{array}{lccccccc}
\overline{z}_2 = \max \frac{59}{7} & & -\frac{4}{7}x_3 & -\frac{1}{7}x_4 & & & & \\
x_1 & & +\frac{1}{7}x_3 & +\frac{2}{7}x_4 & & & = & \frac{20}{7} \\
& x_2 & & +x_4 & & & = & 3 \\
& & -\frac{2}{7}x_3 & +\frac{10}{7}x_4 & +x_5 & & = & \frac{23}{7} \\
& & \frac{1}{7}x_3 & +\frac{2}{7}x_4 & & +t & = & -\frac{1}{7} \\
x_1, & x_2, & x_3, & x_4, & x_5, & t & \geq & 0
\end{array}$$

is infeasible, $\overline{z}_2 = -\infty$, and hence node S_2 is *pruned by infeasibility.*

Choosing a Node. The node list now contains S_{11}, S_{12}. Arbitrarily choosing S_{12}, we remove it from the list.

Reoptimizing. $S_{12} = S \cap \{x : x_1 \leq 2, x_2 \geq 1\}$. The resulting linear program has optimal solution $\overline{x}^{12} = (2, 1)$ with value 7. As $\overline{x}^{12}$ is integer, $z^{12} = 7$.

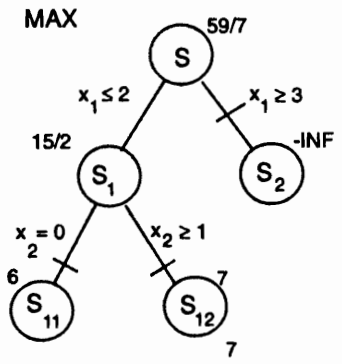

Fig. 7.8 Complete branch and bound tree

Updating the Incumbent. As the solution of $LP(S_{12})$ is integer, we update the value of the best feasible solution found $\underline{z} \leftarrow \max\{\underline{z}, 7\}$, and store the corresponding solution $(2, 1)$. S_{12} is now *pruned by optimality.*

Choosing a Node. The node list now contains only S_{11}.

Reoptimizing. $S_{11} = S \cap \{x : x_1 \leq 2, x_2 \leq 0\}$. The resulting linear program has optimal solution $\overline{x}^{11} = (\frac{3}{2}, 0)$ with value 6. As $\underline{z} = 7 > \overline{z}_{11} = 6$, the node is *pruned by bound.*

Choosing a Node. As the node list is empty, the algorithm terminates. The incumbent solution $x = (2, 1)$ with value $z = 7$ is optimal.

The complete branch-and-bound tree is shown in Figure 7.8. In Figure 7.9 we show graphically the feasible node sets S_i, the branching, the relaxations $LP(S_i)$, and the solutions encountered in the example.

7.4 LP-BASED BRANCH AND BOUND

In Figure 7.10 we present a flowchart of a simple branch and bound algorithm, and then discuss in more detail some of the practical aspects of developing and using such an algorithm.

Storing the Tree. In practice one does not store a tree, but just the list of *active* nodes or subproblems that have not been pruned and that still need to be explored further. Here the question arises of how much information one should keep. Should one keep a minimum of information and be prepared to repeat certain calculations, or should one keep all the information available? At a minimum, the best known dual bound and the variable lower and upper

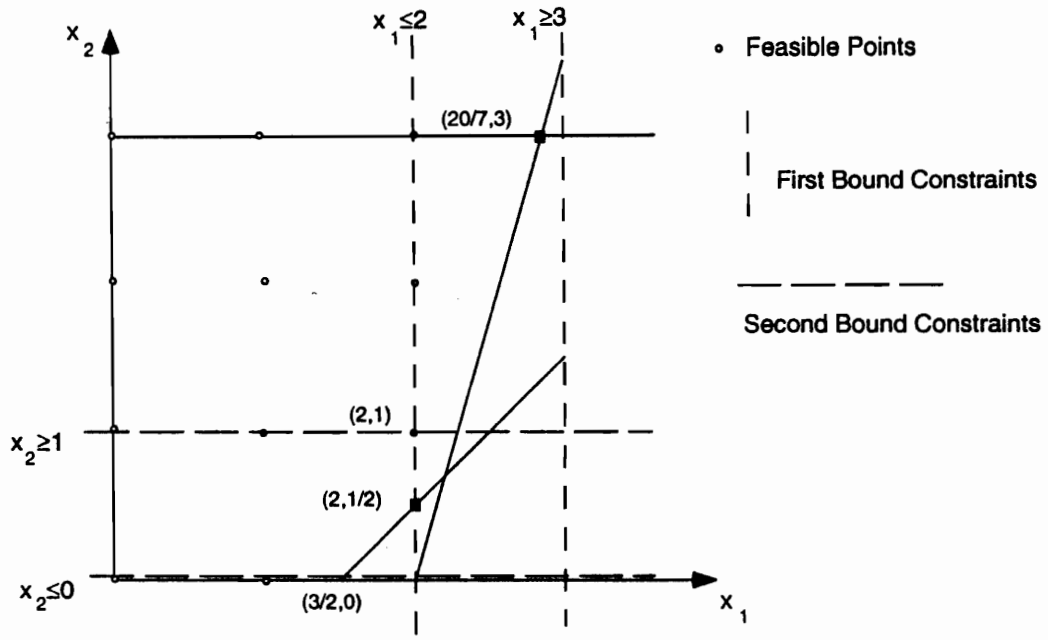

Fig. 7.9 Division of the feasible region

bounds needed to restore the subproblem are stored. Usually one also keeps an optimal or near-optimal basis, so that the linear programming relaxation can be reoptimized rapidly.

Returning to the questions raised earlier, there is no single answer that is best for all instances. One needs to use rules based on a combination of theory, common sense, and practical experimentation. In our example, the question of **how to bound** was solved by using an LP relaxation; **how to branch** was solved by choosing an integer variable that is fractional in the LP solution. However, as there is typically a choice of a set C of several candidates, we need a rule to choose between them. One common choice is *the most fractional variable*:

$$\arg \max_{j \in C} \min[f_j, 1 - f_j]$$

where $f_j = x_j^* - \lfloor x_j^* \rfloor$, so that a variable with fractional value $f_j = \frac{1}{2}$ is best. Other rules are based on the idea of *estimating* the cost of forcing the variable x_j to become integer.

How to choose a node was avoided by making an arbitrary choice. In practice there are several contradictory arguments that can be invoked:

(i) It is only possible to prune the tree significantly with a (primal) feasible solution, giving a hopefully good lower bound. Therefore one should descend as quickly as possible in the enumeration tree to find a first feasible solution. This suggests the use of a *Depth-First Search* strategy. Another argument for

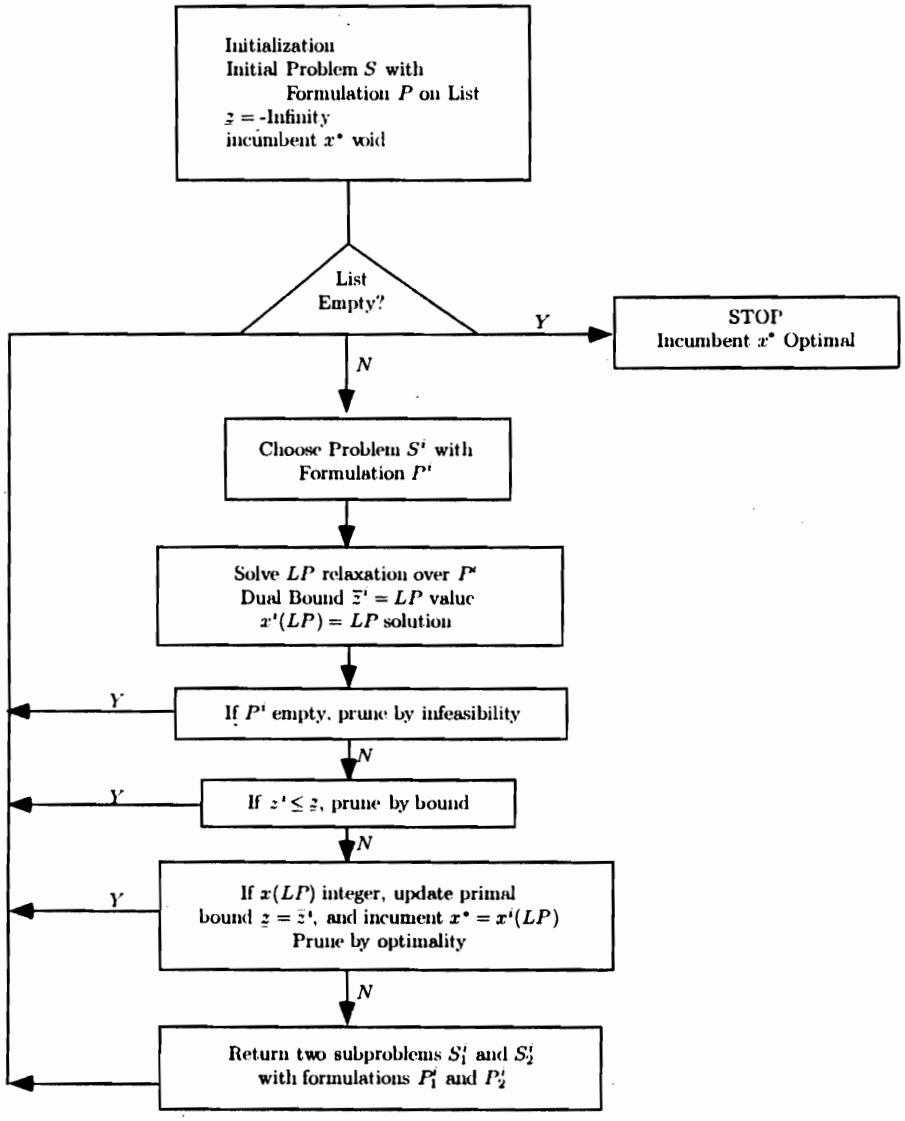

Fig. 7.10 Branch-and-bound flow chart

such a strategy is the observation that it is always easy to resolve the linear programming relaxation when a simple constraint is added, and the optimal basis is available. Therefore passing from a node to one of its immediate descendants is to be encouraged. In the example this would imply that after treating node S_1, the next node treated would be S_{11} or S_{12} rather than S_2.

(ii) To minimize the total number of nodes evaluated in the tree, the optimal strategy is to always choose the active node with the best (largest upper) bound (i.e., choose node s where $\overline{z}_s = \max_t \overline{z}_t$). With such a rule, one will never divide any node whose upper bound $\overline{z}_t$ is less than the optimal value z. This leads to a *Best-Node First* strategy. In the example of the previous section, this would imply that after treating node S_1, the next node chosen

would be S_2 with bound $\frac{59}{7}$ from its predecessor, rather than S_{11} or S_{12} with bound $\frac{15}{2}$.

In practice a compromise between these ideas is often adopted, involving an initial depth-first strategy until at least one feasible solution has been found, followed by a strategy mixing best node and depth first so as to try to prove optimality and also find better feasible solutions.

7.5 USING A BRANCH-AND-BOUND SYSTEM

Commercial branch-and-bound systems for integer and mixed integer programming are essentially as described in the previous section, and the default strategies have been chosen by tuning over hundreds of different problem instances. The basic philosophy is to solve and resolve the linear programming relaxations as rapidly as possible, and if possible to branch intelligently. Given this philosophy, all recent systems contain, or offer,

1. A powerful (automatic) preprocessor, which simplifies the model by reducing the number of constraints and variables, so that the linear programs are easier
2. The simplex algorithm with a choice of pivoting strategies, and an interior point option for solving the linear programs
3. Limited choice of branching and node selection options
4. Use of priorities

and some offer

5. GUB/SOS branching
6. Strong branching
7. Reduced cost fixing
8. Primal heuristics

In this section we briefly discuss those topics requiring user intervention. Preprocessing, which is very important, but automatic, is presented in the (optional) next section. Reduced cost fixing is treated in Exercise 7.7, and primal heuristics are discussed in Chapter 12.

Priorities. Priorities allow the user to tell the system the relative importance of the integer variables. The user provides a file specifying a value (importance) of each integer variable. When it has to decide on a branching variable, the system will choose the highest priority integer variable whose current linear programming value is fractional. At the same time the user can specify a preferred branching direction telling the system which of the two branches to

explore first.

GUB Branching. Many models contain generalized upper bound (GUB) or special ordered sets (SOS) of the form

$$\sum_{j=1}^{k} x_j = 1$$

with $x_j \in \{0,1\}$ for $j = 1,\ldots,k$. If the linear programming solution x^* has some of the variables $x_1^*,\ldots,x_k^*$ fractional, then the standard branching rule is to impose $S_1 = S \cap \{x : x_j = 0\}$ and $S_2 = S \cap \{x : x_j = 1\}$ for some $j \in \{1,\ldots,k\}$. However, because of the GUB constraint, $\{x : x_j = 0\}$ leaves $k-1$ possibilities $\{x : x_i = 1\}_{i \neq j}$ whereas $\{x : x_j = 1\}$ leaves only one possibility. So S_1 is typically a much larger set than S_2, and the tree is *unbalanced.*

GUB branching is designed to provide a more balanced division of S into S_1 and S_2. Specifically the user specifies an ordering of the variables in the GUB set $j_1,\ldots,j_k$, and the branching scheme is then to set

$$S_1 = S \cap \{x : x_{j_i} = 0 \; i = 1,\ldots r\} \text{ and}$$
$$S_2 = S \cap \{x : x_{j_i} = 0 \; i = r+1,\ldots k\},$$

where $r = \min\{t : \sum_{i=1}^{t} x_{j_i}^* \geq \frac{1}{2}\}$. In many cases such a branching scheme is much more effective than the standard scheme, and the number of nodes in the tree is significantly reduced.

User Options (a) **Cutoffs.** If the user knows or can construct a good feasible solution to his or her problem, it is very important that its value is passed to the system as the incumbent value to serve as a *cutoff* in the branch and bound.

(b) **Simplex Strategies.** Though the linear programming algorithms are finely tuned, the default strategy will not be best for all classes of problems. Different *simplex pricing* strategies may make a huge difference in running times for a given class of models, so if similar models are resolved repeatedly or the linear programs seem very slow, some experimentation by the user with pricing strategies is permitted. In addition, on very large models, *interior point methods* may be best for the solution of the first linear program. Unfortunately, up to now such methods are still not good for reoptimizing quickly at each node of the tree.

(c) **Strong Branching.** The idea behind strong branching is that on difficult problems it should be worthwhile to do more work to try to choose a better branching variable. The system chooses a set C of basic integer variables that are fractional in the LP solution, branches up and down on each of them in

turn, and reoptimizes on each branch either to optimality, or for a specified number of dual simplex pivots. Now for each variable $j \in C$, it has upper bounds z_j^D for the down branch and z_j^U for the up branch. The variable having the largest effect (decrease of the dual bound)

$$j^* = \arg\ \min_{j \in C} \max[z_j^D, z_j^U]$$

is then chosen, and branching really takes place on this variable. Obviously, solving two LPs for each variable in C is costly, so such branching should only be used when the other criteria have been found to be ineffective.

7.5.1 If All Else Fails

What can one do if a particular problem instance turns out to be difficult, meaning that after a certain time

(i) no feasible solution has been found, or

(ii) the gap between the value of the best feasible solution and the value of the dual upper bound is unsatisfactorily large, or

(iii) the system runs out of space because there are too many active nodes in the node list?

Finding Feasible Solutions. This is in general $\mathcal{NP}$-hard. Some systems have simple primal heuristics embedded in them. Also as discussed earlier, using priorities and directions for branching can help. How to find feasible solutions, starting from the LP solution or using explicit problem structure, is the topic of Chapter 12.

Finding Better Dual Bounds. Branch-and-bound algorithms fail very often because the bounds obtained from the linear programming relaxations are too weak. This means that tightening up the formulation of the problem is of crucial importance. Systematic ways to do this are the subject of Chapters 8–11. Specifically the addition of constraints or cuts to improve the formulation is treated in Chapers 8 and 9, leading to the development of a potentially more powerful branch-and-cut algorithm. The Lagrangian relaxation and column generation approaches of Chapters 10 and 11 provide alternative ways to strengthen the formulation by convexifying part of the feasible region.

7.6 PREPROCESSING*

Before solving a linear or integer program, it is natural to check that the formulation is "sensible", and as strong as possible given the information available.

All the commercial branch-and-bound systems carry out such a check, called *preprocessing*. The basic idea is to try to quickly detect and eliminate redundant constraints and variables, and tighten bounds where possible. Then if the resulting linear/integer program is smaller/tighter, it will typically be solved much more quickly. This is especially important in the case of branch-and-bound because tens or hundreds of thousands of linear programs may need to be solved.

First we demonstrate linear programming preprocessing by an example.

Example 7.4 Consider the linear program

$$\begin{array}{lrcrcrcl} \max & 2x_1 & + & x_2 & - & x_3 & & \\ & 5x_1 & - & 2x_2 & + & 8x_3 & \leq & 15 \\ & 8x_1 & + & 3x_2 & - & x_3 & \geq & 9 \\ & x_1 & + & x_2 & + & x_3 & \leq & 6 \\ & 0 & \leq & x_1 & \leq & 3 & & \\ & 0 & \leq & x_2 & \leq & 1 & & \\ & 1 & \leq & x_3. & & & & \end{array}$$

Tightening Bounds. Isolating variable x_1 in the first constraint we obtain

$$5x_1 \leq 15 + 2x_2 - 8x_3 \leq 15 + 2 \times 1 - 8 \times 1 = 9$$

where we use the bound inequalities $x_2 \leq 1$ and $-x_3 \leq -1$. Thus we obtain the tightened bound $x_1 \leq \frac{9}{5}$.

Similarly isolating variable x_3, we obtain

$$8x_3 \leq 15 + 2x_2 - 5x_1 \leq 15 + 2 \times 1 - 5 \times 0 = 17,$$

and the tightened bound $x_3 \leq \frac{17}{8}$.

Isolating variable x_2, we obtain

$$2x_2 \geq 5x_1 + 8x_3 - 15 \geq 5 \times 0 + 8 \times 1 - 15 = -7.$$

Here the existing bound $x_2 \geq 0$ is not changed.

Turning to the second constraint, isolating x_1 and using the same approach, we obtain $8x_1 \geq 9 - 3x_2 + x_3 \geq 9 - 3 + 1 = 7$, and an improved lower bound $x_1 \geq \frac{7}{8}$.

No more bounds are changed based on the second or third constraints. However, as certain bounds have been tightened, it is worth passing through the constraints again.

Constraint 1 for x_3 now gives $8x_3 \leq 15 + 2x_2 - 5x_1 \leq 15 + 2 - 5 \times \frac{7}{8} = \frac{101}{8}$. Thus we have the new bound $x_3 \leq \frac{101}{64}$.

Redundant Constraints. Using the latest upper bounds in constraint 3, we see that

$$x_1 + x_2 + x_3 \leq \frac{9}{5} + 1 + \frac{101}{64} < 6,$$

and so this constraint is redundant and can be discarded. The problem is now reduced to

$$\begin{array}{lcccc} \max & 2x_1 & +x_2 & -x_3 & \\ & 5x_1 & -2x_2 & +8x_3 & \leq 15 \\ & 8x_1 & +3x_2 & -x_3 & \geq 9 \\ & \frac{7}{8} \leq x_1 & \leq \frac{9}{5}, & 0 \leq x_2 \leq 1, & 1 \leq x_3 \leq \frac{101}{64}. \end{array}$$

Variable Fixing (by Duality). Considering variable x_2, observe that increasing its value makes all the constraints (other than its bound constraints) less tight. As the variable has a positive objective coefficent, it is advantageous to make the variable as large as possible, and thus set it to its upper bound of 1. (Another way to arrive at a similar conclusion is to write out the LP dual. For the dual constraint corresponding to the primal variable x_2 to be feasible, the dual variable associated with the constraint $x_2 \leq 1$ must be positive. This implies by complemementary slackness that $x_2 = 1$ in any optimal solution.)

Similarly, decreasing x_3 makes the constraints less tight. As the variable has a negative objective coefficient, it is best to make it as small as possible, and thus set it to its lower bound $x_3 = 1$. Finally the LP is reduced to the trivial problem

$$\max\{2x_1 : \tfrac{7}{8} \leq x_1 \leq \tfrac{9}{5}\}.$$ ∎

Formalizing the above ideas is straightforward.

Proposition 7.3 *Consider the set* $S = \{x : a_0x_0 + \sum_{j=1}^{n} a_jx_j \leq b, l_j \leq x_j \leq u_j \text{ for } j = 0, 1, \ldots, n\}$.

(i) Bounds on Variables. If $a_0 > 0$, *then*

$$x_0 \leq (b - \sum_{j:a_j>0} a_jl_j - \sum_{j:a_j<0} a_ju_j)/a_0,$$

and if $a_0 < 0$, *then*

$$x_0 \geq (b - \sum_{j:a_j>0} a_jl_j - \sum_{j:a_j<0} a_ju_j)/a_0.$$

(ii) Redundancy. The constraint $a_0x_0 + \sum_{j=1}^{n} a_jx_j \leq b$ *is redundant if*

$$\sum_{j:a_j>0} a_ju_j + \sum_{j:a_j<0} a_jl_j \leq b.$$

(iii) Infeasibility. $S = \emptyset$ *if*

$$\sum_{j:a_j>0} a_jl_j + \sum_{j:a_j<0} a_ju_j > b.$$

(iv) Variable Fixing. For a maximization problem in the form: $\max\{cx : Ax \leq b, l \leq x \leq u\}$, *if* $a_{ij} \geq 0$ *for all* $i = 1, \ldots, m$ *and* $c_j < 0$, *then* $x_j = l_j$. *Conversely if* $a_{ij} \leq 0$ *for all* $i = 1, \ldots, m$ *and* $c_j > 0$, *then* $x_j = u_j$.

Turning now to integer programming problems, preprocessing can sometimes be taken a step further. Obviously, if $x_j \in Z^1$ and the bounds l_j or u_j are not integer, we can tighten to

$$\lceil l_j \rceil \leq x_j \leq \lfloor u_j \rfloor.$$

For mixed integer programs with variable upper and lower bound constraints $l_j y_j \leq x_j \leq u_j y_j$ with $y_j \in \{0,1\}$, it is also important to use the tightest bound information.

For $BIPs$ it is common to look for simple "logical" or "boolean" constraints involving only one or two variables, and then either add them to the problem or use them to fix some variables. Again we demonstrate by example.

Example 7.5 Consider the set of constraints involving four 0–1 variables:

$$\begin{array}{rrrrcr} 7x_1 & +3x_2 & -4x_3 & -2x_4 & \leq & 1 \\ -2x_1 & +7x_2 & +3x_3 & +x_4 & \leq & 6 \\ & -2x_2 & -3x_3 & -6x_4 & \leq & -5 \\ 3x_1 & & -2x_3 & & \geq & -1 \\ & & x & \in & B^4. & \end{array}$$

Generating Logical Inequalities. Examining row 1, we see that if $x_1 = 1$, then necessarily $x_3 = 1$, and similarly $x_1 = 1$ implies $x_4 = 1$. This can be formulated with the linear inequalities $x_1 \leq x_3$ and $x_1 \leq x_4$. We see also that the constraint is infeasible if both $x_1 = x_2 = 1$ leading to the constraint $x_1 + x_2 \leq 1$.

Row 2 gives the inequalities $x_2 \leq x_1$ and $x_2 + x_3 \leq 1$.

Row 3 gives $x_2 + x_4 \geq 1$ and $x_3 + x_4 \geq 1$.

Row 4 gives $x_1 \geq x_3$.

Combining Pairs of Logical Inequalities. We consider pairs involving the same variables.

From rows 1 and 4, we have $x_1 \leq x_3$ and $x_1 \geq x_3$, which together give $x_1 = x_3$.

From rows 1 and 2, we have $x_1 + x_2 \leq 1$ and $x_2 \leq x_1$ which together give $x_2 = 0$. Now from $x_2 + x_4 \geq 1$ and $x_2 = 0$, we obtain $x_4 = 1$.

Simplifying. Making the substitutions $x_2 = 0, x_3 = x_1, x_4 = 1$, all four constraints of the feasible region are redundant, and we are left with $x_1 \in \{0,1\}$, so the only feasible solutions are $(1,0,1,1)$ and $(0,0,0,1)$. ∎