

# Contents

|          |  |            |
|----------|--|------------|
| <b>8</b> | <b>The Branch and Bound Approach</b>   | <b>375</b> |
| 8.1      | The Difference Between Linear<br>and Integer Programming Models . . . . .          | 375        |
| 8.2      | The Three Main Tools in the Branch and Bound Approach                              | 377        |
| 8.3      | The Strategies Needed to Apply the Branch and Bound<br>Approach . . . . .          | 380        |
| 8.3.1    | The Lower Bounding Strategy . . . . .  | 381        |
| 8.3.2    | The Branching Strategy . . . . .   | 382        |
| 8.3.3    | The Search Strategy . . . . .  | 385        |
| 8.4      | The 0–1 Knapsack Problem . . . . .   | 393        |
| 8.5      | The General MIP . . . . .  | 405        |
| 8.6      | B&B Approach for Pure 0–1 IPs . . . . .  | 409        |
| 8.7      | Advantages and Limitations of the B&B Approach, Re-<br>cent Developments . . . . . | 417        |
| 8.8      | Exercises . . . . .  | 420        |
| 8.9      | References . . . . .   | 423        |





# Chapter 8

## The Branch and Bound Approach

This is Chapter 8 of “Junior Level Web-Book for Optimization Models for decision Making” by Katta G. Murty.

### 8.1 The Difference Between Linear and Integer Programming Models

The algorithms that we discussed in earlier chapters for linear programs, and some recently developed algorithms such as interior point methods not discussed in this book, are able to solve very large scale LP models arising in real world applications within reasonable times (i.e., within a few hours of time on modern supercomputers for truly large models). This has made linear programming a highly viable practical tool. If a problem can be modeled as an LP with all the data in it available, then we can expect to solve it and use the solution for decision making; given adequate resources such as computer facilities and a good software package, which are becoming very widely available everywhere these days.

Unfortunately, the situation is not that rosy for integer and com-

binatorial optimization models. The research effort devoted to these areas is substantial, and it has produced very fundamental and elegant theory, but has not delivered algorithms on which practitioners can place faith that exact optimum solutions for large scale models can be obtained within reasonable times.

Certain types of problems, like the knapsack problem, and the traveling salesman problem (TSP), seem easier to handle than others. Knapsack problems involving 10,000 or more 0–1 variables and TSPs involving a few thousands of cities, have been solved very successfully in at most a few hours of computer time on modern parallel processing supercomputers by implementations of branch and bound methods discussed in this chapter custom-made to solve them using their special structure. But for many other types of problems discussed in Chapter 7, only moderate sized problems may be solvable to optimality within these times by existing techniques. Real world applications sometimes lead to large scale problems. When faced with such problems, practitioners usually resort to heuristic methods which may obtain good solutions in general, but cannot guarantee that they will be optimal. We discuss some of these heuristic methods in Chapter 9.

The main theoretical differences between linear programs, and the discrete optimization problems discussed in Chapter 7 are summarized below.

Linear programs    There are theoretically proven necessary and sufficient optimality conditions which can be used to check efficiently whether a given feasible solution is an optimum solution or not (these are existence of a dual feasible solution that satisfies the complementary slackness optimality conditions together with the given primal feasible solution). These optimality conditions have been used to develop algebraic methods such as the simplex method and other methods for solving LPs.

Discrete and combinatorial optimization problems

For these problems discussed in Chapter 7, there are no known optimality conditions to check whether a given feasible solution is optimal, other than comparing this solution with every other feasible solution implicitly or explicitly. That is why discrete optimization problems are solved by enumerative methods that search for the optimum solution in the set of feasible solutions.

## 8.2 The Three Main Tools in the Branch and Bound Approach

The total enumeration method presented in Chapter 7 evaluates every feasible solution to the problem and selects the best. This method is fine for solving small problems for which the number of solutions is small. But for large scale real world applications, the total enumeration method is impractical as the number of solutions to evaluate is very large.

Branch and bound is an approach to search for an optimum feasible solution by doing only a **partial enumeration**. The Branch and Bound approach was developed independently in the context of the traveling salesman problem (TSP) in [K. G. Murty, C. Karel, and J. D. C. Little, 1962], and in the context of integer programming in [A. H. Land and A. G. Doig, 1960]. Particularly, the important concept of bounding is from the former reference.

We will use the abbreviation “B&B” for “Branch and Bound”. We will describe the main principles behind the B&B approach using a problem in which an objective function  $z(x)$  is to be minimized (as before, a problem in which an objective function  $z'(x)$  is to be maximized is handled through the equivalent problem of minimizing  $-z'(x)$  subject to the same constraints). Let  $\mathbf{K}_0$  denote the set of feasible solutions of the original problem, and  $z_0$  the unknown optimum objective value in it. The main tools that the B&B approach uses to solve this problem are the following.

**Branching or Partitioning** In the course of applying the B&B approach,  $\mathbf{K}_0$  is partitioned into many simpler subsets. This is what one would do in practice if one is looking, say, for a needle in a haystack. The haystack is big and it is impossible to search all of it simultaneously. So, one divides it visually into approximately its right and left halves, and selects one of the halves to search for the needle first, while keeping the other half aside to be pursued later if necessary.

Each subset in the partition of  $\mathbf{K}_0$  will be the set of feasible solutions of a problem called a **candidate problem** abbreviated as “CP”, which is the original problem augmented by additional constraints called **branching constraints** generated by the branching operation. This subset is actually stored by storing the CP, i.e., essentially storing the branching constraints in that CP.

In each stage, one promising subset in the partition is chosen and an effort made to find the best feasible solution from it. If the best feasible solution in that subset is found, or if it is discovered that the subset is empty (which happens if the corresponding CP is infeasible), we say that the associated CP is **fathomed**. If it is not fathomed, that subset may again be partitioned into two or more simpler subsets (this is the **branching operation**) and the same process repeated on them.

**Bounding** The B&B approach computes and uses both upper and lower bounds for the optimum objective value.

The upper bound  $u$ , of which there is only one at any stage, is always an upper bound for the unknown  $z_0$ , the minimum objective value in the original problem. It is always the objective value at some known feasible solution. To find an upper bound one finds a feasible solution  $\bar{x}$  (preferably one with an objective value close to the minimum) and takes  $z(\bar{x})$  as the upper bound. When there are constraints, it may be difficult to find a feasible solution satisfying them. In that case we will not have an upper bound at the beginning of the algorithm, but the moment a feasible solution is produced in the algorithm we will begin to have an upper bound.

At any stage,  $u$ , the current upper bound for  $z_0$  is the least among the objective values of all the feasible solutions that turned up in the algorithm so far. The feasible solution whose objective value is the

current upper bound  $u$  is called **the incumbent** at that stage. Thus the incumbent and the upper bound change whenever a better feasible solution appears during the algorithm.

In contrast to the upper bound of which there is only one at any stage, each candidate problem has its own separate lower bound for the minimum objective value among feasible solutions of that CP. For any CP,

**Lower bound for a CP** = it is a number  $\ell$  computed by a procedure called **the lower bounding strategy**, satisfying the property that every feasible solution for this CP has objective value  $\geq \ell$ .

**Pruning** Suppose we have an upper bound  $u$  for the unknown  $z_0$  at some stage. Any CP associated with a lower bound  $\ell \geq u$  has the property that all its feasible solutions have objective value  $\geq u =$  objective value of the current incumbent, so none of them is better than the current incumbent. In this case the algorithm **prunes** that CP, i.e., discards its set of feasible solutions from further consideration.

As an example, suppose  $\bar{x}$  is the current incumbent (i.e., the best feasible solution to the original problem known so far) with an objective value of 30. Since the objective function  $z(x)$  is to be minimized, we know that the optimum objective value in the original problem  $z_0$  is  $\leq 30$ , so  $u = 30$  is the current upper bound for the unknown  $z_0$ . If the lower bound for the minimum objective value in a CP, say CP1 is 32, we can prune CP1, because every feasible solution for CP1 has objective value  $\geq 32$ , and the current incumbent  $\bar{x}$  is better than all of them.

**High Quality Lower Bounds** Consider a particular CP, say CP1. Let  $z_1$  denote the unknown minimum value for the objective function  $z(x)$  among feasible solutions of CP1. Suppose we are able to compute a lower bound for  $z_1$  by two different methods, say Method1, and Method2.

Suppose Method1 gives 25 as a lower bound for  $z_1$  (i.e., it concludes that  $z_1 \geq 25$ ); and Method2 gives 32 as the lower bound for  $z_1$ . Here both the methods are correct, but the information obtained by Method2 that  $z_1 \geq 32$  is more valuable than the information given by Method1. That's why the quality of a lower bound (for the optimum objective value  $z_1$  in this problem CP1) is judged by how large it is, the larger the lower bound the higher its quality.

There are usually many different strategies that one can use to compute a lower bound for  $z_1$ . The computational effort needed for them, and the values of the lower bound produced by them for  $z_1$  may be different. Once an incumbent for the problem is obtained, if the lower bounding strategy used on each CP produces a **high quality lower bound** (i.e., a lower bound as high as possible, or close to the minimum objective value in this CP), then a lot of pruning may take place, thereby curtailing enumeration.

Thus the bounding step in the B&B approach contributes significantly to the efficiency of the search for an optimum solution of the original problem, particularly if the lower bounding strategy used produces high quality lower bounds without too much computational effort. The lower bounds are used in selecting promising CPs to pursue in the search for the optimum, and in pruning CPs whose set of feasible solutions cannot possibly contain a better solution than the current incumbent. Also, the lower bounding strategy applied on a CP may produce fortuitously the best feasible solution for it, thus fathoming it.

### 8.3 The Strategies Needed to Apply the Branch and Bound Approach

As before, we consider a problem in which an objective function  $z(x)$  is to be minimized subject to a given system of constraints on the decision variables. We denote the set of feasible solutions of the original problem by  $\mathbf{K}_0$ , and the unknown minimum value of  $z(x)$  in the original problem by  $z_0$ .

### 8.3.1 The Lower Bounding Strategy

$z_0$ , the minimum value of  $z(x)$ , is obtained precisely if the original problem is solved, but it may be hard to solve. The purpose of applying the lower bounding strategy on the original problem is to compute a lower bound for  $z_0$ , i.e., a number  $\ell$  satisfying  $\ell \leq z(x)$  for all feasible solutions  $x$  of the original problem. It should be relatively easy to implement and computationally very efficient. Among several lower bounding strategies, the one which gives a bound closest to the minimum objective value without too much computational effort, is likely to make the B&B approach most efficient. Thus in designing a lower bounding strategy, we need to strike a balance between

the quality of the lower bound obtained (the larger the better)

the computational effort involved (the lesser the better)

#### Lower Bounding by Solving a Relaxed Problem

There are several principles that can be used for constructing lower bounding strategies, but we will only discuss the most important one in this book. It is based on solving a **relaxed problem**.

In the lower bounding strategy based on relaxation, we identify the **hard** or **difficult constraints** in the problem. A subset of constraints is said to be hard if there is an efficient algorithm to solve the remaining problem after deleting these constraints. We select one such set, and relax the constraints in it. The remaining problem is called the **relaxed problem**. Since the relaxed problem has fewer (or less restrictive) constraints than the original, its set of feasible solutions contains  $\mathbf{K}_0$  inside it. Hence the minimum objective value in the relaxed problem is a lower bound for the minimum objective value in the original problem.

As an example suppose we want to solve an integer program. Suppose we have a software package to solve linear programs very efficiently. Unfortunately, it cannot be used to solve the integer program we need to solve, since it cannot enforce the condition the the variables must take only integer values. In this case we consider the integer constraints on the variables as the hard constraints in the original problem,

if we relax them we get what is known as the **LP relaxation** of the original integer program. This LP relaxation can be solved by our LP software package, and the minimum objective value in it will be a lower bound for the minimum objective value in the original integer program.

The importance of computing good bounds for the minimum objective value in a combinatorial minimization problem, particularly the lower bound, cannot be overemphasized. In the branch and bound approach, computing a good lower bound is an essential component without which the approach degenerates into total enumeration and will be almost impractical on large scale problems.

Let  $\bar{x}$  be the optimum solution of the relaxed problem. It is known as the initial **relaxed optimum**. Its objective value,  $z(\bar{x})$ , is the lower bound for  $z_0$  obtained by the lower bounding strategy.

If  $\bar{x}$  satisfies the hard constraints that were relaxed, it is feasible to the original problem, and since  $z(x) \geq z(\bar{x})$  for all feasible solutions  $x$  of the original problem,  $\bar{x}$  is an optimum solution for the original problem. If this happens we say that the original problem is **fathomed**, and terminate. Otherwise, the algorithm now applies the branching strategy on the original problem.

### 8.3.2 The Branching Strategy

The branching strategy partitions the set of feasible solutions of the original problem into two or more subsets. Each subset in the partition is the set of feasible solutions of a problem obtained by imposing additional simple constraints called the **branching constraints** on the original problem. These problems are called **candidate problems**.

#### Branching Using a 0–1 Variable

If there is a 0–1 variable,  $x_1$  say, in the problem, we can generate two candidate problems by adding the constraint “ $x_1 = 0$ ” to the original problem for one of them, and “ $x_1 = 1$ ” for the other. Clearly the sets of feasible solutions of the two CPs generated are disjoint, and since  $x_1$  is required to be either 0 or 1 in every feasible solution of the original problem, the union of the sets of feasible solutions of the two

CPs is  $\mathbf{K}_0$ . Thus this branching operation partitions  $\mathbf{K}_0$  into the sets of feasible solutions of the two CPs generated.

### Branching Using a Nonnegative Integer Variable

Suppose there is a variable,  $x_2$  say, in the problem which is a nonnegative integer variable, and the value of  $x_2$  in the relaxed optimum is 6.4 say (in general assume it is  $\bar{x}_2$ ). Then we can generate two candidate problems by adding the constraint “ $x_2 \leq 6$ ” (in general “ $x_2 \leq \lfloor \bar{x}_2 \rfloor$ ”) to the original problem for one of them, and “ $x_2 \geq 7$ ” (in general “ $x_2 \geq \lfloor \bar{x}_2 \rfloor + 1$ ”) for the other. Here again, clearly the set of feasible solutions of the two CPs is a partition of  $\mathbf{K}_0$ .

The variables  $x_1, x_2$  used in the branching operations described above are known as the **branching variables** for those operations.

### Applying Lower Bounding Strategy on Each New CP Generated

Now the lower bounding strategy is applied on each CP generated. The constraints relaxed in the CP for lower bounding will always be the hard constraints from the system of constraints in the original problem; the branching constraints in the CP are never relaxed for lower bounding because these constraints are usually simple constraints that can be handled easily by algorithms used to solve the relaxed problem.

$\hat{x}$ , the computed optimum solution of the relaxed problem used for getting a lower bound for the minimum objective value in that CP, is known as the **relaxed optimum** for that CP.

Then  $z(\hat{x})$  is a lower bound for the minimum objective value in that CP. If  $\hat{x}$  satisfies all the relaxed constraints (i.e., it is a feasible solution for the CP), then by the argument made earlier,  $\hat{x}$  is in fact an optimum solution for this CP, and hence  $z(\hat{x})$  is the true minimum objective value in this CP. If this happens we say that this CP is **fathomed**.

In general, a CP is said to be fathomed whenever we find a feasible solution  $\hat{x}$  for it with objective value equal to the computed lower bound for the minimum objective value in this CP, then  $\hat{x}$  is an optimum solution for that CP.

If this is the first CP to be fathomed,  $\hat{x}$  becomes the **first incumbent**, and  $z(\hat{x})$  the current upper bound for the unknown  $z_0$ , the minimum objective value in the original problem. If this is not the first CP to be fathomed,  $\hat{x}$  replaces the present incumbent to become the new incumbent if  $z(\hat{x}) <$  the present upper bound for  $z_0$ ; and  $z(\hat{x})$  becomes the new upper bound for  $z_0$  (this operation is called **updating the incumbent**). Otherwise (i.e., if  $z(\hat{x}) \geq$  the present upper bound for  $z_0$ ), there is no change in the incumbent and the CP is **pruned**. Thus, the **current incumbent** at any stage is the best feasible solution obtained so far, and its objective value is the **current upper bound** for the unknown  $z_0$ .

If a candidate problem,  $P$  say, is not fathomed, we only have a lower bound for the minimum objective value in it. When this candidate problem  $P$  is pursued next, the branching strategy will be applied on it to generate two candidate subproblems such that the following properties hold.

1. Each candidate subproblem is obtained by imposing additional branching constraints on the candidate problem  $P$ .
2. The sets of feasible solutions of the candidate subproblems form a partition of the set of feasible solutions of the candidate problem  $P$ .
3. The lower bounds for the minimum objective values in the candidate subproblems are as high as possible.

The operation of generating the candidate subproblems is called **branching the candidate problem  $P$** . The candidate problem  $P$  is known as the **parent problem** for the candidate subproblems generated; and the candidate subproblems are the **children** of  $P$ . The important thing to remember is that a candidate subproblem always has all the constraints in the original problem, and all the branching constraints of its parent, and the branching constraints added to the system by the branching operation which created it. Thus every candidate subproblem inherits all the constraints of its parent, plus the branching constraint just introduced. Hence the lower bound for the

minimum objective value in any candidate subproblem will be  $\geq$  the lower bound associated with its parent.

### How to Select the Branching Variable Among Several Available?

When there are several variables which can be selected as the branching variable, it should be selected among them so as to satisfy Property 3 above as best as possible to increase the overall efficiency of the algorithm.

The branching strategy must provide good selection criteria for branching variables, to achieve this goal. One way this is done is to compute an estimate for the difference between the lower bound for the most important among the candidate subproblems generated, and the lower bound for the parent, if a particular eligible variable is selected as the branching variable at that stage. An estimate like that is known as an **evaluation coefficient** for that variable. Then the branching variable can be selected to be the variable with the highest evaluation coefficient among the eligible variables.

### 8.3.3 The Search Strategy

Initially the original problem is the only candidate problem. Begin by applying the lower bounding strategy on it. Let  $x^0$  be the relaxed optimum solution obtained, and  $L_0$  the lower bound for  $z_0$ . If  $x^0$  satisfies the hard constraints that were relaxed, it is an optimum solution for the original problem which is fathomed in this case, and the algorithm terminates. If  $x^0$  violates some of the relaxed constraints, the situation at this stage can be represented as in Figure 8.1.

Now apply the branching strategy on the original problem, generating candidate problems CP 1 and CP 2. Apply the lower bounding strategy on these CPs and enter them as in Figure 10.2. This diagram is known as the **search tree** at this stage. The original problem has already been branched, and CP 1, CP 2 are its children. Nodes CP 1, CP 2 which have not yet been branched and hence have no children yet,

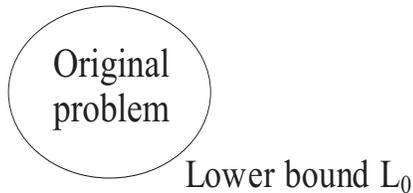


Figure 8.1:

are known as **terminal nodes** (also called **live nodes** in the search tree at this stage.

At any stage of the algorithm, the **list** (also called **stack** in some books, but we will use the word stack in a slightly different sense when using backtrack search strategy discussed later) denotes the collection of all the unfathomed and unpruned CPs which are terminal nodes at that stage.

If either of CP 1, CP 2 is fathomed, the optimum solution in it becomes the incumbent and its objective value the upper bound  $u$  for the unknown  $z_0$ . There is no reason to pursue a fathomed CP, hence it is deleted from the list. If both CP1, CP 2 are fathomed, the best of their optimum solutions is an optimum solution of the original problem, and the algorithm terminates.

Suppose  $L_1 \leq L_2$ . If CP 1 is fathomed, then  $L_1 =$  current upper bound = objective value of the current incumbent which is the optimum solution for CP 1. In this case the lower bound  $L_2$  for CP 2 is  $\geq$  the objective value of the incumbent, and hence CP 2 is pruned; and the algorithm terminates again with the incumbent as the optimum solution for the original problem.

If CP 2 is fathomed but not CP 1 and  $L_1 < L_2$ , then we cannot prune CP 1 since it may contain a feasible solution better than the present incumbent. In this case CP 1 joins the list and it will be branched next.

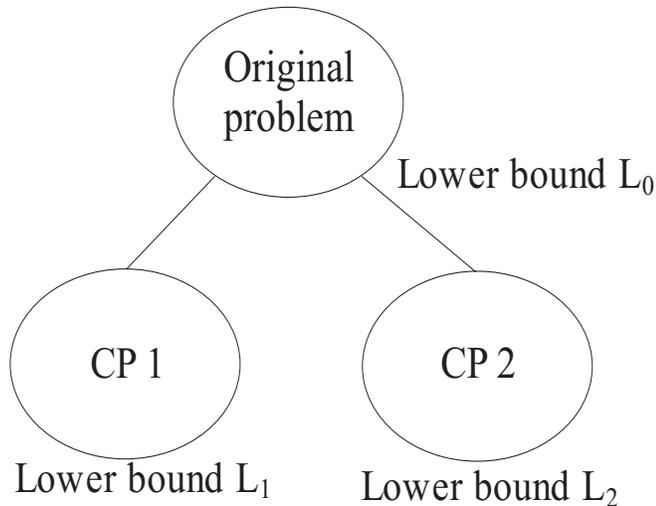


Figure 8.2:

Suppose both CP 1 and CP 2 are unfathomed and  $L_1 < L_2$ . Now there is a possibility that the optimum solution for CP 1 has an objective value  $< L_2$ , and if so it will be optimal to the original problem. Thus at this stage both CP 1 and CP 2 are in the list, but CP 1 is branched next, while CP 2 is left in the list to be pursued later if necessary. Any CP which is not branched yet, not fathomed and not pruned, is known as a **live node** in the search tree at this stage. It is a terminal node which is in the list.

When CP 1 is branched, suppose the candidate subproblems CP 11, CP 12 are generated. The search tree at this stage is shown in Figure 8.3. CP 1 is no longer a terminal node, so it is deleted from the list.

Now the lower bounding strategy is applied on the new CPs, CP 11, CP 12. Any CP whose relaxed problem is infeasible cannot have any feasible solution, and so is pruned. If any of these CPs is fathomed, update the incumbent and the upper bound for  $z_0$ . The incumbent at any stage is the best (i.e., the one with the least objective value) among feasible solutions of the original problem identified at the various occurrences of fathoming so far, and its objective value is the current

upper bound for the minimum objective value in the original problem. Whenever a newly generated CP is fathomed, it is never added to the list, the optimum solution in it is used to update the incumbent. Any CP whose lower bound is  $\geq$  the upper bound for  $z_0$ , is pruned and taken off the list. Therefore at any stage of the algorithm, the list (or stack) consists of all the unpruned, unfathomed, and unbranched CPs at that stage. The following properties will hold.

- (i) The sets of feasible solutions of the CPs in the list (or stack) are mutually disjoint.
- (ii) If there is an incumbent at this stage, any feasible solution of the original problem that is strictly better than the current incumbent is a feasible solution of some CP in the list.

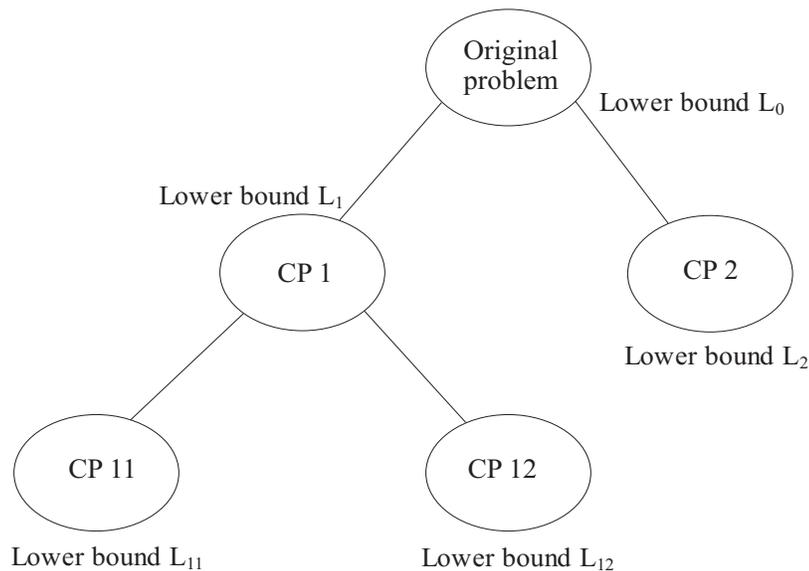


Figure 8.3:

- (iii) If there is no incumbent at this stage, the union of the sets of feasible solutions of the CPs in the list is the set of feasible solutions of the original problem.

In a general stage, identify a CP that is associated with the least lower bound among all CPs in the list at this stage. Denote this CP by  $P$ . Delete  $P$  from the list and apply the branching strategy on it. Apply the lower bounding strategy on the candidate subproblems generated. If any of them turn out to be infeasible, prune them. If any of them is fathomed, update the incumbent. If there is a change in the incumbent, look through the list and prune. Add the unpruned and unfathomed among the newly generated candidate subproblems to the list. Then go to the next stage.

At the stage depicted in Figure 8.3, CP 2, CP 11, CP 12 are in the list. If  $L_2 = \min\{L_2, L_{11}, L_{12}\}$ , then CP 2 is branched next producing CP 21, CP 22 say with lower bounds  $L_{21}, L_{22}$  respectively, leading to the search tree in Figure 8.4. The search trees are drawn in this discussion to illustrate how the search is progressing. In practice, the algorithm can be operated with the list of CPs, and the incumbent when it is obtained, and updating these after each stage.

### Criterion for Selecting the Next CP From List to Branch

In general the search strategy specifies the sequence in which the generated CPs will be branched. We discussed the search strategy which always selects the next CP to be branched, to be the one associated with the **least lower bound** among all the CPs in the list at that stage. It is a **priority strategy** with the least lower bound as the priority criterion. Some people call it a **jump-track strategy** because it always jumps over the list looking for the node with the least lower bound to branch next. This search strategy with the least lower bound criterion seems to be an excellent strategy that helps to minimize the total number of nodes branched before the termination of the algorithm.

The algorithm terminates when the list of CPs becomes empty. At termination if there is an incumbent, it is an optimum feasible solution of the original problem. If there is no incumbent at termination, the original problem is infeasible.

Another search strategy that is popular in computer science appli-

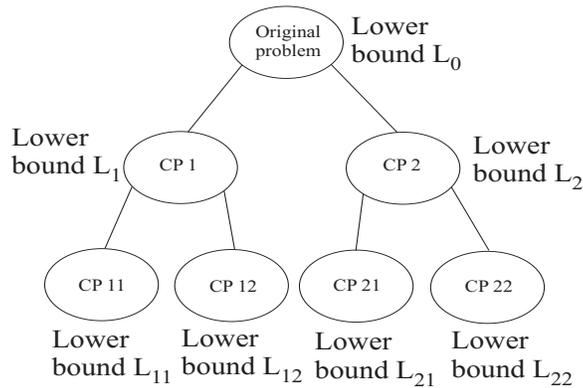


Figure 8.4:

cations is the **backtrack search strategy** based on depth-first search. It keeps one of the CPs from the list for the purpose of the search and calls it the **current candidate problem** or **current CP**. The other CPs in the list constitute the **stack**.

If the current CP is fathomed, the incumbent is updated and the current CP is discarded. If the incumbent changes, the necessary pruning is carried out in the stack. Then a CP from the stack is selected as the new current CP (the selection criterion is discussed below), and the algorithm is continued.

If the current CP is not fathomed, the branching strategy is applied on it and the lower bounding strategy applied on the candidate subproblems generated. If both these candidate subproblems are fathomed, the incumbent is updated, pruning is carried out in the stack, and a new current CP is selected from the stack (see below for the selection criterion). If only one of the candidate subproblems generated is fathomed, the incumbent is updated, pruning is carried out in the stack, and if the other candidate subproblem is unpruned it is made the new current CP. If neither of the subproblems generated is fathomed, the more promising one among them (may be the one associated with the least lower bound among them) is made the new current CP, the other candidate subproblem is added to the stack and the algorithm is

continued.

## How to Select a Current CP From the Stack in Backtrack

In backtrack search, whenever the current CP is fathomed, or after branching the current CP if the two child CPs generated are fathomed or pruned, a new current CP has to be selected from the stack to continue the algorithm.

When the algorithm has to select a current CP from the stack, the best selection criteria seems to be that of choosing the most recent CP added to the stack. This selection criterion is called **LIFO (Last In First Out)**.

If backtrack search strategy is employed, the algorithm terminates when it is necessary to select a CP from the stack, and the stack is found empty at that time. If there is an incumbent at that stage, it is an optimum solution of the original problem. If there is no incumbent at that stage, the original problem is infeasible.

In either search strategy, if a CP is sufficiently small that it is practical to search for an optimum solution for it by total enumeration, it is better to do it than to continue branching it further.

## How to Make B&B Approach Efficient?

For the B&B approach to work well, the bounding strategy must provide a lower bound fairly close to the minimum objective value in the problem but with little computational effort. The branching strategy must generate candidate subproblems that have lower bounds as high as possible.

A well designed B&B algorithm makes it possible to do extensive and effective pruning throughout, and thus enables location of the optimum by examining only a small fraction of the overall set of feasible solutions. That is why B&B methods are known as **partial enumeration methods**.

Practical experience indicates that the search strategy based on the least lower bound allows for extensive pruning, and hence leads to more

efficient algorithms.

In most well designed B&B algorithms, it often happens that an optimum feasible solution of the original problem is obtained as an incumbent at an early stage, but the method goes through a lot of computation afterwards to confirm its optimality. That is why a good heuristic to use on large scale problems is to terminate the algorithm when the limit on available computer time is reached, and take the current incumbent as a near optimum solution.

We will now formally state the basic step in the B&B approach to solve a problem. First, a lower bounding strategy, a branching strategy, and a search strategy have to be developed for the problem. If the problem size is large, good lower bounding and branching strategies are very critical to the overall efficiency of the algorithm; and almost always these strategies have to be tailormade for the problem to exploit its special nature, structure and geometry. Once these strategies are developed, the algorithm proceeds as follows.

#### THE BRANCH AND BOUND ALGORITHM

**Initialization** Apply the lower bounding strategy on the original problem and compute a lower bound for the minimum objective value. If the original problem is fathomed, we have an optimum solution, terminate. If the relaxed problem used for lower bounding is infeasible, the original problem is infeasible too, terminate. If neither of these occur, put the original problem in the list and go to the general step.

**General Step** If the list has no CPs in it; the original problem is infeasible if there is no incumbent at this stage; otherwise the current incumbent is an optimum solution for it. Terminate.

If the list is nonempty, use the search strategy to retrieve a CP from it for branching next. Apply the branching strategy on the selected CP, and apply the lower bounding strategy on each of the candidate subproblems generated at branching. Prune or discard any of them that turn out to be infeasible; and if any of them are fathomed, update the incumbent and the upper bound for the minimum. Any candidate subproblem, or CP in the list whose

lower bound is  $\geq$  the present upper bound is now pruned. Add the unfathomed and unpruned candidate subproblems to the list, and go to the next step.

The application of the B&B approach will now be illustrated with some examples.

## 8.4 The 0–1 Knapsack Problem

We consider the 0–1 knapsack problem in this section. As described in Chapter 7, in this problem there are  $n$  objects which can be loaded into a knapsack whose capacity by weight is  $w_0$  weight units. For  $j = 1$  to  $n$ , object  $j$  has weight  $w_j$  weight units, and value  $v_j$  money units. Only one copy of each object is available to be loaded into the knapsack. None of the objects can be broken; i.e., each object should be either loaded whole into the knapsack, or should be left out. The problem is to decide the subset of objects to be loaded into the knapsack so as to maximize the total value of the objects included, subject to the weight capacity of the knapsack. So, defining for  $j = 1$  to  $n$

$$x_j = \begin{cases} 1, & \text{if } j\text{th article is packed into the knapsack} \\ 0, & \text{otherwise} \end{cases}$$

the problem is

$$\begin{aligned} \text{Minimize } z(x) &= - \sum_{j=1}^n v_j x_j \\ \text{subject to } \sum_{j=1}^n w_j x_j &\leq w_0 \end{aligned} \quad (8.4.1)$$

$$0 \leq x_j \leq 1 \quad \text{for all } j$$

$$x_j \text{ integer} \quad \text{for all } j \quad (8.4.2)$$

Here the objective function  $z(x)$  is the negative total value of the objects loaded into the knapsack, it states the objective function in minimization form.

## Eliminating Objects Heavier than Knapsack Capacity

If there is an object  $j$  such that  $w_j > w_0$ , it cannot enter the knapsack because its weight exceeds the knapsack's weight capacity. For all such objects  $j$ ,  $x_j = 0$  in every feasible solution of (8.4.1), (8.4.2). Identify all such objects and fix all the corresponding variables at 0 and delete them from further consideration. To solve the problem, we need only find the values of the remaining variables  $x_j$  satisfying  $w_j \leq w_0$ , in an optimum solution.

The remaining problem (8.4.1) is an LP; and so if we relax the integer requirements (8.4.2), we can solve the remaining problem by efficient LP methods. The lower bounding strategy based on relaxing the integer requirements on the variables is called the **LP relaxation strategy**. We will use it. Because of its special structure (only one constraint, and all the variables are subject to finite lower and upper bounds), the relaxed LP (8.4.1) can be solved very efficiently by the following special procedure. The objective value of the optimum solution of the relaxed LP is a lower bound for the minimum objective value in the original problem.

## Special Procedure for Solving the LP Relaxation of the 0–1 Knapsack Problem

Suppose the knapsack's weight capacity is  $w_0$  weight units; and there are  $n$  objects available for loading into it, with the  $j$ th object having weight  $w_j$  weight units and value  $v_j$  money units, for  $j = 1$  to  $n$ .

In the LP relaxation, variables are allowed to take fractional values. Since it is an LP with only one equality constraint, it can be solved by a special algorithm (different from the simplex method discussed earlier for general LPs) which is very efficient, we describe this special algorithm now.

To find the optimum solution of the LP relaxation of the 0–1 knapsack problem, first fix all variables  $x_j$  corresponding to  $j$  satisfying  $w_j > w_0$  at 0 and remove them from further consideration.

Then compute the **density** (value per unit weight,  $d_j = v_j/w_j$  for

object  $j$ ) of each remaining object, and arrange the objects in decreasing order of this density from top to bottom. Begin making  $x_j = 1$  from the top in this order until the weight capacity of the knapsack is reached, at that stage make the last variable equal to a fraction until the weight capacity is completely used up; and make all the remaining variables equal to 0.

### Example 8.4.1:

Consider the journal subscription problem discussed in Section 7.2. The various journals are the objects in it, the subscription price of the journal plays the role of its weight, and the readership of the journal plays the role of its value. Here is the data for the problem from Section 7.2, with the objects arranged in decreasing order of density from top to bottom.

| *Object $j$ | *Weight $w_j$ | *value $v_j$ | *Density | *Cumulative |
|-------------|---------------|--------------|----------|-------------|
| 1           | 80            | 7840         | 98       | 80          |
| 8           | 99            | 8316         | 84       | 179         |
| 4           | 165           | 15015        | 74       | 344         |
| 3           | 115           | 8510         | 74       | 459         |
| 2           | 95            | 6175         | 65       | 554         |
| 5           | 125           | 7375         | 59       | 679         |
| 6           | 78            | 1794         | 23       | 757         |
| 7           | 69            | 897          | 13       | 826         |

\* Object = journal, weight = annual subscription  
value = annual readership, density =  $v_j/w_j$ ,  
cumulative = total weight upto this object

From Section 7.2, the available budget for annual subscription to these journals, \$670 =  $w_0$  plays the role of the knapsack's capacity by weight in this example, and all objects have weight  $< w_0$ . In the last column of the table we provided the cumulative total weight of all the objects from the top and up to (including) that object. We begin loading objects into the knapsack (here it can be interpreted as

renewing the subscriptions) from the top. By the time we come to object number 2, \$554 of the knapsack's capacity is used up, leaving \$670 - 554 = 116. The next journal, object 5, has a subscription price of \$125, the money left in the budget at this stage covers only 116/125 of this journal's subscription. So, the optimum solution of the LP relaxation of this example problem is  $\hat{x} = (\hat{x}_1, \hat{x}_8, \hat{x}_4, \hat{x}_3, \hat{x}_2, \hat{x}_5, \hat{x}_6, \hat{x}_7) = (1, 1, 1, 1, 1, 116/125, 0, 0)$ . Or, arranging the variables in serial order of subscripts, and as a column vector it is  $\hat{x} = (\hat{x}_1, \text{to } \hat{x}_8) = (1, 1, 1, 1, 116/125, 0, 0, 1)^T$ .

## Fathoming Strategy

If the optimum solution,  $\hat{x}$ , of the LP relaxation is integral (i.e., every variable has a value of 0 or 1 in it), then that solution  $\hat{x}$  is an optimum solution of the original 0–1 problem, and thus the original problem is fathomed.

## The Branching Strategy

From the procedure described above, it is clear that if the optimum solution for the LP relaxation,  $\hat{x}$ , is not integral, there will be exactly one variable which has a fractional value in it. Suppose it is  $\hat{x}_p$ . A convenient branching strategy is to select  $x_p$  as the branching variable and generate two CPs, CP 1 (CP 2) by including the branching constraint " $x_p = 0$ " (" $x_p = 1$ ") over those of the original problem. Since  $\hat{x}_p$  is fractional, this branching strategy eliminates the present LP relaxed optimum  $\hat{x}$  from further consideration as it is not feasible to either CP 1 or CP 2. We will use this branching strategy because it identifies the branching variable unambiguously, and its property of eliminating the current LP relaxed optimum from further consideration is quite nice.

The branching constraints in a general CP, say CP N, in this algorithm will be of the following form.

$$\begin{aligned} x_{q_1} = x_{q_2} = \dots = x_{q_r} &= 0 \\ x_{p_1} = x_{p_2} = \dots = x_{p_u} &= 1 \end{aligned} \tag{8.4.3}$$

CP N is the original problem with these branching constraints as additional constraints. The  $r + u$  variables  $x_{q_1}, \dots, x_{q_r}, x_{p_1}, \dots, x_{p_u}$  are called **fixed variables** in this CP N because their values are fixed in it by the branching constraints.

In the same way every CP obtained in this algorithm will fix a subset of variables at 0, and another subset of variables at 1. And the sum of the weights of the variables fixed at 1 in any CP will always be  $\leq$  the knapsack's weight capacity, as otherwise the CP will have no feasible solution.

In CP N, objects  $p_1, \dots, p_u$  are required to be included in the knapsack, and objects  $q_1, \dots, q_r$  are required to be excluded from it by the branching constraints. So, we only have  $w_0^N = w_0 - (w_{p_1} + \dots + w_{p_u})$  of the knapsack's weight capacity left to be considered in CP N; and objects in  $\Gamma_N = \{1, \dots, n\} \setminus \{q_1, \dots, q_r, p_1, \dots, p_u\}$  available to load. Any object  $j \in \Gamma_N$  whose weight  $w_j$  is  $> w_0^N =$  remaining knapsack capacity, cannot be included in the knapsack in this CP; hence the corresponding variable  $x_j$  must be fixed at 0 and removed from further consideration in this CP. We assume that these constraints are already included in (8.4.3).

All variables  $x_j$  for  $j \in \Gamma_N$  which are not fixed in CP N, are called **free variables** in this CP, since they are free to assume values of 0 or 1 in feasible solutions of this CP. So, the remaining problem in CP N is a smaller knapsack problem with choice restricted to objects in  $\Gamma_N$  and knapsack's weight capacity equal to  $w_0^N$ . It is the following problem.

$$\begin{aligned} \text{Minimize} \quad & - \sum_{j \in \Gamma_N} v_j x_j \\ \text{subject to} \quad & \sum_{j \in \Gamma_N} w_j x_j \leq w_0^N \\ & x_j = 0 \text{ or } 1 \text{ for all } j \end{aligned} \tag{8.4.4}$$

So, to get a lower bound for the minimum objective value in CP N, we need to solve the LP relaxation of (8.4.4), for which the special procedure discussed earlier can be used. When the optimum solution for the LP relaxation of (8.4.4) is combined with the values of the

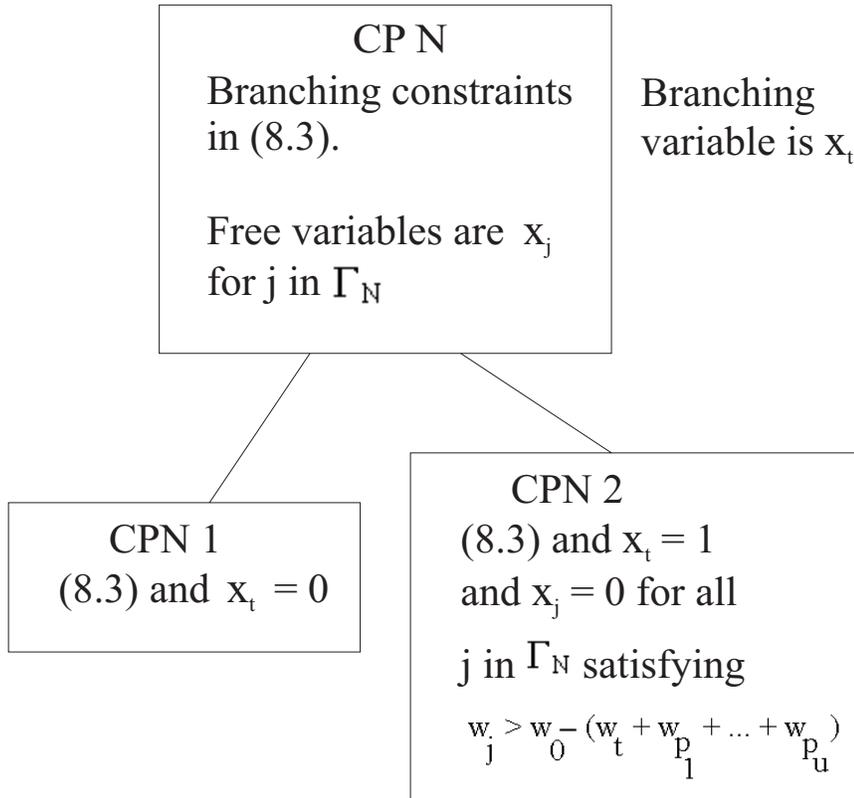


Figure 8.5: Candidate problems generated when CP N is branched using  $x_t$  as the branching variable.

fixed variables in the branching constraints in (8.4.3) in this CP N, we get the LP relaxed optimum,  $\bar{x}$  say, for CP N; and its objective value is a lower bound for the minimum objective value in CP N. If  $\bar{x}$  is integral, it is an optimum solution for CP N, and in this case CP N is fathomed. If this happens, we update the incumbent, remove CP N from the list, and select a new CP from the list to branch next and continue the algorithm. If  $\bar{x}$  is not integral, there will be a unique variable which has a fractional value in it, suppose it is  $x_t$ . When CP N is to be branched, we will choose  $x_t$  as the branching variable. This generates two CPs as shown in Figure 8.5.

Now the lower bounding strategy is applied on each of CPN 1, CPN 2, and the method is continued.

### Fathoming a CP With Small Number of Free Variables by Enumeration

Consider CP N defined by the branching constraints (8.4.3). The number of free variables in it is  $s = n - r - u$ . The remaining problem in CP N, (8.4.4), is to decide which of the remaining free objects in  $\Gamma_N$  to load into the remaining part of the knapsack with residual capacity  $w_0^N$ . Since  $|\Gamma_N| = s$ , the optimum solution in this CP can be determined by evaluating each of the  $2^s$  subsets of  $\Gamma_N$  to see which are feasible to (8.4.4), and selecting the best among those feasible. This becomes practical if  $s$  is small. Thus if  $s$  is small, we find the optimum solution of CP N by this enumeration instead of continuing to branch it. This is appropriately called **fathoming the CP by enumeration**.

**Example 8.4.2:** For a numerical example we consider a knapsack problem in which the knapsack's weight capacity is  $w_0 = 35$  weight units. There are 9 objects available for loading into the knapsack with data given in the following table.

| Object $j$ | Weight $w_j$ | Value $v_j$ | Density $d_j = v_j/w_j$ |
|------------|--------------|-------------|-------------------------|
| 1          | 3            | 21          | 7                       |
| 2          | 4            | 24          | 6                       |
| 3          | 3            | 12          | 4                       |
| 4          | 21           | 168         | 8                       |
| 5          | 15           | 135         | 9                       |
| 6          | 13           | 26          | 2                       |
| 7          | 16           | 192         | 12                      |
| 8          | 20           | 200         | 10                      |
| 9          | 40           | 800         | 20                      |

Define the decision variables as:

$$x_j = \begin{cases} 1, & \text{if } j\text{th article is packed into the knapsack} \\ 0, & \text{otherwise} \end{cases}$$

Here is the problem.

$$\begin{aligned}
 \text{Minimize } z(x) &= -21x_1 - 24x_2 - 12x_3 - 168x_4 && -135x_5 - \\
 & && 26x_6 - 192x_7 - 200x_8 - 800x_9 \\
 \text{subject to } & 3x_1 + 4x_2 + 3x_3 + 21x_4 && +15x_5 + \\
 & 13x_6 + 16x_7 + 20x_8 + 40x_9 &\leq & 35 && (8.4.5) \\
 & 0 \leq x_j \leq 1 && \text{for all } j
 \end{aligned}$$

$$x_j \text{ integer for all } j \quad (8.4.6)$$

We fix  $x_9 = 0$  because  $w_9 = 40 > w_0 = 35$ , and remove object 9 from further consideration. We need to find the values of the remaining variables  $x_1$  to  $x_8$  in an optimum solution with  $x_9$  fixed at 0. The lower bounding strategy relaxes (8.4.6) and solves the LP relaxation (8.4.5) with  $x_9$  fixed at 0. The densities of the objects are given in the last column in the above tableau. Using the procedure discussed above we find that the LP relaxed optimum is  $x = (x_1 \text{ to } x_9) = (0, 0, 0, 0, 0, 0, 1, 19/20, 0)^T$  with an objective value of  $-382$ . Since  $x_8$  is not integral in this solution, the original problem is not fathomed. A lower bound for the minimum objective value in the original problem is  $-382$ .

Now the original problem has to be branched. As discussed above, we use the variable  $x_8$  with a fractional value in the LP relaxed optimum as the branching variable. CP 1, CP 2 with branching constraints " $x_8 = 0$ ", " $x_8 = 1$ " respectively are generated. In CP 2 object 8 is already loaded into the knapsack, which leaves only 15 weight units of residual capacity in it. Hence object 7 with a weight of 16 cannot fit into the knapsack in CP 2. Thus in CP 2 " $x_7 = 0$ " is an implied branching constraint (the constraint " $x_8 = 1$ " implies " $x_7 = 0$ " in this problem).

The entire search tree for the algorithm is shown in Figure 8.6. The branching constraints in each CP are recorded inside the node representing that CP. Besides each node the LP relaxed optimum for it is recorded by giving the values of the variables that are nonzero in this

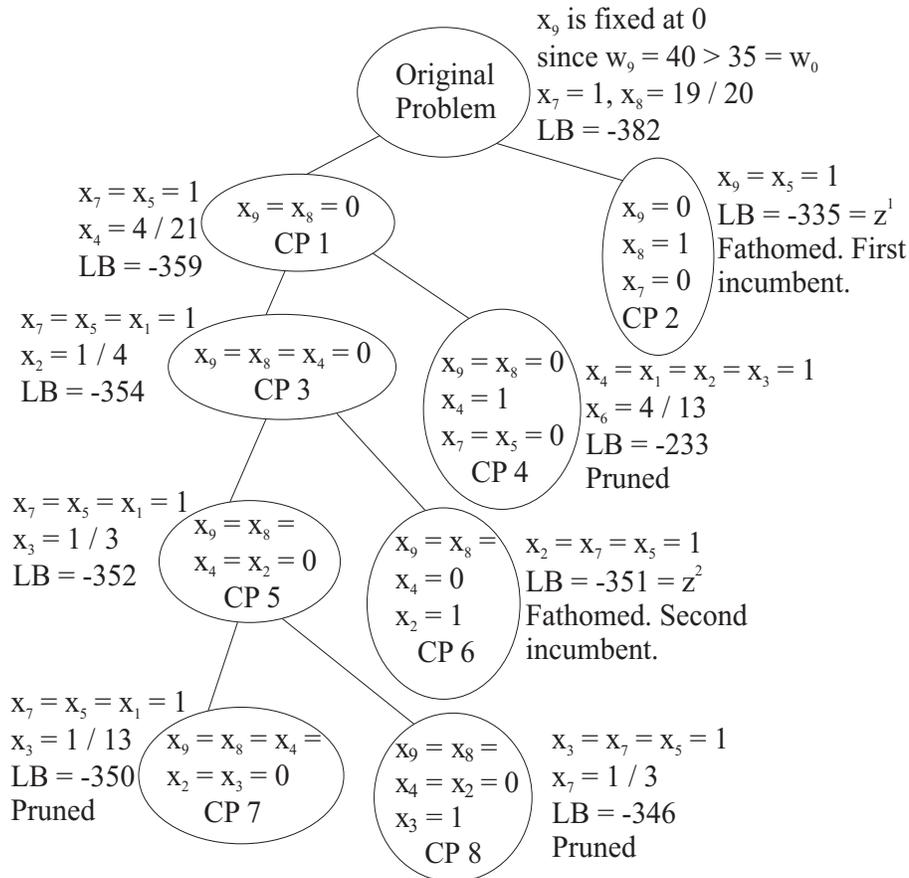


Figure 8.6:

solution. The following abbreviations are used: LB = lower bound, BV = branching variable used for branching.

Here is an explanation of the various stages in the algorithm.

CP 2 is fathomed since the relaxed LP optimum for it is integral. This solution  $x^1 = (0, 0, 0, 0, 1, 0, 0, 1, 0)^T$  is the first incumbent, and its objective value,  $z^1 = -335$ , is the present upper bound for the minimum objective value in the original problem.

Now CP 1 is the only CP in the list, so it is branched next.  $x_4$ , the fractional variable in its relaxed optimum, is used as the BV. This

branching generates CP 3, CP 4. In CP 4  $x_4$  is fixed at 1, and  $x_8, x_9$  are fixed at 0. So, this is a knapsack problem with residual capacity of  $35 - 21 = 14$ , and since  $w_5, w_7$  are both  $> 14$ , we need to set  $x_5 = x_7 = 0$  also as branching constraints in this CP, CP 4. And since the lower bound for CP 4,  $-233$ , is  $>$  present upper bound of  $-335$ , it is pruned.

Now CP 3 is the only CP in the list, so it is branched next, resulting in CP 5, CP 6. CP 6 is fathomed, and the integral relaxed LP optimum in it,  $x^2 = (0, 1, 0, 0, 1, 0, 1, 0, 0)^T$  replaces the present incumbent  $x^1$  as the next incumbent since its objective value  $z^2 = -351 < z^1$ .  $z^2$  is the new upper bound for the minimum objective value in the original problem.

CP 5, the only CP in the list now is branched next, resulting in CP 7, CP 8. Both these are pruned since their lower bounds are  $> z^2$ . The list is now empty, so the present incumbent  $x^2 = (0, 1, 0, 0, 1, 0, 1, 0, 0)^T$  is an optimum solution for the original knapsack problem. This implies that an optimum choice to load into the knapsack is objects 2, 5, and 7, yielding a maximum value loaded of 351, and using up all the 35 units of weight capacity.  $\bowtie$

This is the basic B&B approach for the 0–1 knapsack problem. Recently, several simple mathematical tests have been developed to check whether a given CP in this algorithm has a feasible solution whose objective value is strictly better than that of the current incumbent. If one of these tests indicates that a CP cannot have a feasible solution better than the current incumbent, then the CP is pruned right away. These tests are simple and computationally inexpensive. By implementing such tests we can expect extensive pruning to take place during the algorithm, making the enumeration efficient. With a battery of such tests, modern software packages are able to solve practical 0–1 knapsack problems involving thousands of variables in a few minutes of computer time.

## The Greedy Heuristic for the 0–1 Knapsack Problem

As mentioned above, high quality software is available for solving

large scale 0–1 knapsack problems. However, some practitioners are often reluctant to use such sophisticated techniques to solve their problems, preferring to obtain a near optimum solution by simple heuristic methods instead. The data in their models may not be very reliable, and may contain errors of unknown magnitudes. Or, the true data in the real problem may be subject to random fluctuations, and their model may have been constructed using numbers that represent the best educated guess about their expected values. In such situations, a global optimum solution for the model with the current data may not actually be an optimum solution for the real problem. Investing money to acquire a sophisticated but possibly expensive software package to solve the model with approximate data may not be worthwhile in these situations. So, they reason that it is better to obtain a near optimum solution for the model using a simple heuristic technique.

The most popular among the simple heuristic methods for the 0–1 knapsack problem is the **greedy heuristic** which selects objects for inclusion in the knapsack using the density as the criterion to be greedy upon. It proceeds this way.

Consider the problem involving  $n$  objects with  $w_j, v_j, d_j = v_j/w_j$  as the weight, value, density respectively of object  $j$  for  $j = 1$  to  $n$ ; and  $w_0$  as the knapsack's weight capacity. It first sets all  $x_j$  for  $j$  satisfying  $w_j > w_0$  at value 0. Then it arranges the remaining objects

| Object $j$ | Weight $w_j$ | Value $v_j$ | Density $d_j = v_j/w_j$ |
|------------|--------------|-------------|-------------------------|
| 1          | 15           | 225         | 15                      |
| 2          | 26           | 260         | 10                      |
| 3          | 45           | 495         | 11                      |
| 4          | 10           | 80          | 8                       |
| 5          | 16           | 112         | 7                       |
| 6          | 10           | 60          | 6                       |
| 7          | 6            | 30          | 5                       |

in decreasing order of density from top to bottom. Starting from the top it begins to make  $x_j = 1$  as it goes down until the weight capacity of the knapsack is reached. At some stage if the next object cannot be included in the knapsack because its weight exceeds the remaining

capacity, it makes  $x_j = 0$  for that object; then the process continues with the object below it. It terminates when either the knapsack's weight capacity is used up (in this case,  $x_j$  is made equal to 0 for all objects below the current one), or when all the objects have been examined in this way in decreasing order of density.

As an example consider the 0–1 knapsack problem involving a knapsack of weight capacity 40 weight units, and 7 objects with data given above.

On this problem the greedy method selects the values of the variables in this order:  $x_3 = 0$ ,  $x_1 = 1$ ,  $x_2 = 0$ ,  $x_4 = 1$ ,  $x_5 = 0$ ,  $x_6 = 1$ ,  $x_7 = 0$ ; leading to the solution  $(x_1 \text{ to } x_7) = (1, 0, 0, 1, 0, 1, 0)^T$ .

The greedy method is not guaranteed to produce an optimum solution in general, but usually produces a solution close to the optimum. A mathematical upper bound for the difference between the value of the greedy solution and that of an optimum solution can be derived. For results on these, see [O. H. Ibarra and C. E. Kim, 1975].

## 0–1 Knapsack Problems with Flexible Data

In many applications we encounter 0–1 knapsack models in which slight changes in the value of  $w_0 =$  the knapsack's weight capacity are entirely permissible. An example of this is the journal subscription problem discussed in Example 8.4.1. In this model, the knapsack's weight capacity is the budgeted amount of \$670 for journal subscriptions. The financial VP will be delighted if the librarian wants to decrease this quantity by any amount; also he may not object to small increases in this quantity. If this quantity can be increased to \$679 (a small increase of \$9), then the solution  $(x_1 \text{ to } x_8) = (1, 1, 1, 1, 1, 0, 0, 1)^T$  becomes feasible (this solution is obtained by selecting journals in decreasing order of density, and uses up the budgeted quantity of \$679 exactly) and is an optimum solution for the problem with this modification. Here, it makes sense to argue with the financial VP to agree to this slight modification.

In all such situations where the value of  $w_0$  is flexible, one can look at two solutions to the original problem. One is  $\hat{x}$ , the solution of the original problem obtained by the greedy heuristic. The other is  $\tilde{x}$

obtained by rounding up to 1 the value of the fractional variable in the LP relaxed optimum corresponding to the original problem. If one can increase  $w_0$  to the capacity used by  $\tilde{x}$ , then  $\tilde{x}$  is an optimum solution for the modified problem. If  $w_0$  can be decreased to the capacity used by  $\hat{x}$ , then  $\hat{x}$  is either optimal or near optimal to the modified problem. The decision makers can look at both  $\hat{x}$  and  $\tilde{x}$  and decide which solution is more desirable for the real problem, and make the appropriate change. In this situation, this may be the most appropriate way to handle this problem instead of trying to solve the model with the original value for  $w_0$  to optimality using an expensive B&B package.

## 8.5 B&B Approach for the General MIP

We consider the following general MIP

$$\begin{aligned} \text{Minimize } & z(x, y) = cx + dy \\ \text{subject to } & Ax + Dy = b \\ & x, y \geq 0 \end{aligned} \tag{8.5.1}$$

$$y \text{ integer vector} \tag{8.5.2}$$

If there are no continuous variables  $x$  in the problem, it is a pure IP.

A lower bounding strategy for this problem is to solve the relaxed LP (8.5.1) obtained by relaxing the integer requirements (8.5.2), using LP techniques. If the relaxed LP (8.5.1) is infeasible, the MIP is clearly infeasible too, then prune it and terminate. On the other hand if the relaxed LP has an optimum solution suppose it is  $(x^0, y^0)$  with an objective value of  $z^0$ . If  $y^0$  satisfies the constraints (8.5.1) that were relaxed,  $(x^0, y^0)$  is an optimum solution of the MIP which is now fathomed, terminate. Otherwise,  $z^0$  is a lower bound for the minimum objective value in the MIP.

A convenient branching strategy is to select one of the integer variables  $y_j$  whose value  $y_j^0$ , in the relaxed LP optimum is noninteger, as

the branching variable. If  $y_j$  is a 0–1 variable, generate two CPs by imposing one additional constraint “ $y_j = 0$ ” or “ $y_j = 1$ ” on the original MIP. If  $y_j$  is a general nonnegative integer variable, generate two CPs by imposing one additional constraint “ $y_j \leq \lfloor y_j^0 \rfloor$ ” or “ $y_j \geq 1 + \lfloor y_j^0 \rfloor$ ” respectively on the original MIP. Here  $\lfloor y_j^0 \rfloor$  is what is called “the floor of  $y_j^0$ ”, which is the greatest integer value that is  $\leq y_j^0$ . For example,  $\lfloor 6.2 \rfloor = 6$ , and  $\lfloor -7.2 \rfloor = -8$ .

If there are several  $j$ 's such that  $y_j^0$  is noninteger, the branching variable is selected from them so as to make the lower bounds for the CPs generated after branching as high as possible. The data in the optimum simplex tableau (the optimum dual solution, i.e., the marginal values) can be used to get estimates (called **penalties**) of the amount by which the lower bounds for the CPs are greater than the lower bound of their parent, but this takes us beyond the scope of this book. Interested readers can see [G. L. Nemhauser and L. A. Wolsey, 1988] for a discussion of these penalties and their use in selecting the branching variable.

A consequence of selecting the branching variable among integer variables with fractional values in the relaxed LP optimum  $(x^0, y^0)$  is that this point  $(x^0, y^0)$  is eliminated from further consideration. This is a nice property.

The lower bounds for the newly generated CPs are computed by solving the relaxed LPs obtained by relaxing the integer requirements on the  $y$ 's in them. The relaxed LP corresponding to a CP contains just one additional constraint (the new branching constraint in it) over those in the relaxed LP for the parent problem. From the known relaxed LP optimum of the parent problem, a relaxed LP optimum for the CP can be obtained by using very efficient sensitivity analysis techniques (these techniques to handle the addition of a new constraint are not discussed in this book; see [K. G. Murty, 1983] for them).

A CP is fathomed when the relaxed LP optimum for it satisfies the integer requirements on the  $y$ 's. The moment a CP is fathomed in the algorithm, we have an incumbent. Each time a new CP is fathomed, we update the incumbent. The current upper bound for the minimum objective value in the original MIP is always the objective value of the current incumbent. Any CP in the list whose lower bound is  $\geq$  the

current upper bound is immediately pruned. Also, if the relaxed LP corresponding to a CP is infeasible, so is that CP, and hence that CP is pruned.

CPs for branching are selected from the list by the least lower bound criterion. The algorithm terminates when the list becomes empty. If there is no incumbent at termination, the original MIP is infeasible. Otherwise, the final incumbent is an optimum solution of the original MIP.

**Example:** Consider the following MIP

| Original Tableau: Tableau 1 |       |       |       |       |       |      |     |
|-----------------------------|-------|-------|-------|-------|-------|------|-----|
| $y_1$                       | $y_2$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $-z$ | $b$ |
| 1                           | 0     | 0     | 1     | -2    | 1     | 0    | 3/2 |
| 0                           | 1     | 0     | 2     | 1     | -1    | 0    | 5/2 |
| 0                           | 0     | 1     | -1    | 1     | 1     | 0    | 4   |
| 0                           | 0     | 0     | 3     | 4     | 5     | 1    | -20 |

$y_1, y_2 \geq 0$ , and integer;  $x_1$  to  $x_4 \geq 0$ ;  $z$  to be minimized

We obtain the LP relaxation of this MIP by relaxing the integer requirements on the variables  $y_1, y_2$ . It can be verified that Tableau 1 is already the canonical tableau WRT the basic vector  $(y_1, y_2, x_1)$ , and that it actually satisfies the optimality criterion for the LP relaxation. From it, we find that the optimum solution for the relaxed LP obtained by relaxing the integer requirements on  $y_1, y_2$  is  $(y^0, x^0) = (3/2, 5/2; 4, 0, 0, 0)$ , with an objective value of  $z^0 = 20$ . Since this solution does not satisfy the integer requirements on  $y_1, y_2$  the MIP is not fathomed. It has to be branched.

Both  $y_1, y_2$  have nonintegral values in the relaxed LP optimum solution. We selected  $y_2$  as the branching variable (BV). Branching leads to CP 1, CP 2 shown in Figure 8.7 given below. The constraints inside a node in Figure 8.7 are the additional (branching) constraints in it over those of the original problem. By the side of each node in Figure 8.7 we give the relaxed LP optimum (RO) corresponding to that node.

For example, the relaxed LP for CP 2 is the following, where  $s_1$  is the slack variable =  $y_2 - 3$  for the branching constraint  $y_2 \geq 3$  in it.

| CP 2 with branching constraints: Tableau 2 |       |       |       |       |       |       |      |       |
|--|-------|-------|-------|-------|-------|-------|------|-------|
| $y_1$                                      | $y_2$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $s_1$ | $-z$ | $b$   |
| 1  | 0     | 0     | 1     | -2    | 1     | 0     | 0    | $3/2$ |
| 0  | 1     | 0     | 2     | 1     | -1    | 0     | 0    | $5/2$ |
| 0  | 0     | 1     | -1    | 1     | 1     | 0     | 0    | 4     |
| 0  | 1     | 0     | 0     | 0     | 0     | -1    | 0    | 3     |
| 0  | 0     | 0     | 3     | 4     | 5     | 0     | 1    | -20   |

$y_1, y_2 \geq 0, s_1 \geq 0; x_1$  to  $x_4 \geq 0; z$  to be minimized

Continuing, we get the following search tree.

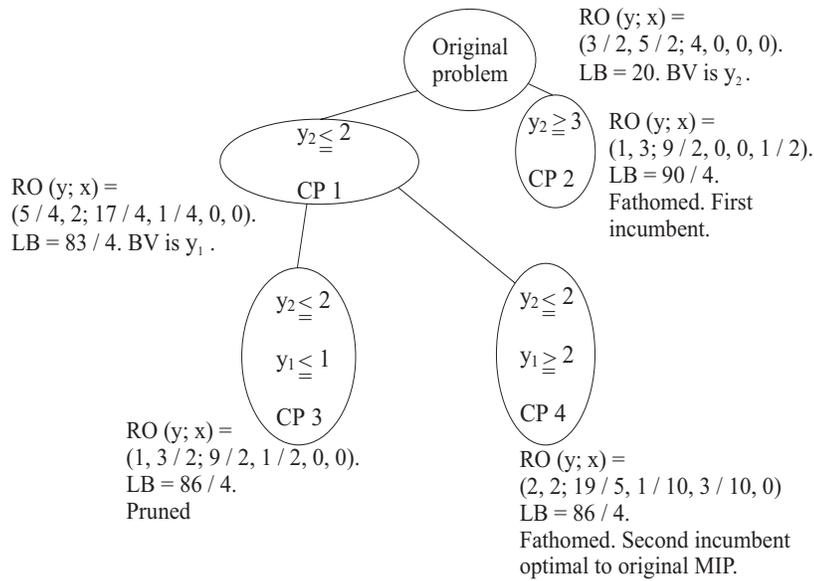


Figure 8.7:

The optimum solution of the original MIP is the second incumbent  $(y; x) = (2, 2; 19/5, 1/10, 3/10, 0)$ , with an objective value of  $86/4$ .

## 8.6 B&B Approach for Pure 0–1 IPs

We consider the problem (8.6.1) where  $A$  is of order  $m \times n$  and  $x \in R^n$ . For some  $j$  if  $c_j$  is  $< 0$ , transform the problem by substituting  $x_j = 1 - y_j$ . In the transformed problem, the objective coefficient of  $y_j$  is  $> 0$ . After similar transformations as necessary, we get a problem of the same form as (8.6.1), but with  $c \geq 0$ . In the rest of the section we assume that  $c \geq 0$ .

$$\begin{aligned} \text{Minimize } & z(x) = cx \\ \text{subject to } & Ax \leq b \\ & x_j = 0 \text{ or } 1 \text{ for all } j \end{aligned} \tag{8.6.1}$$

**Example 8.6.1:** For example, consider the problem with  $n = 4$ , and  $x = (x_1, x_2, x_3, x_4)^T$  as the vector of binary variables in it. Let the objective function to be minimized be  $z(x) = -7x_1 - 5x_2 + 3x_4$ , with negative cost coefficients for  $x_1, x_2$ .

Transform the problem by substituting  $x_1 = 1 - y_1$ ,  $x_2 = 1 - y_2$  wherever  $x_1, x_2$  appear in the problem. The objective function  $z(x)$  becomes  $-7(1 - y_1) - 5(1 - y_2) + 3x_4 = 7y_1 + 5y_2 + 3x_4 - 12$ . So, when expressed in terms of the new variables  $(y_1, y_2, x_3, x_4)^T$ , the objective function to be minimized is  $z'(y_1, y_2, x_3, x_4)^T = 7y_1 + 5y_2 - 12 + 3x_4 - 12$  and  $-12$  being a constant can be dropped from the optimization effort. So, in the modified problem the objective function to be minimized is  $7y_1 + 5y_2 + 3x_4$  with nonnegative cost coefficients for all the variables.

If an optimum solution of the modified problem is  $(\bar{y}_1, \bar{y}_2, \bar{x}_3, \bar{x}_4)^T$ , the corresponding optimum solution of the original problem is  $(\bar{x}_1 = 1 - \bar{y}_1, \bar{x}_2 = 1 - \bar{y}_2, \bar{x}_3, \bar{x}_4)^T$ .

### The Structure of a General CP in this Algorithm

In the B&B algorithm discussed below, CPs are obtained by selecting a subset of the variables  $x_j$  and fixing each of them at value 0 or 1 (these are the branching constraints in this CP). Any variable fixed at

0 (1) is called a **0-variable** (**1-variable**) in that CP. The 0-variables fixed at value 0, and the 1-variables fixed at value 1, constitute what is known as a **partial solution**. Each CP generated in the algorithm corresponds to a partial solution. Variables that are not fixed at 0 or 1 in a CP are called **free variables** in that CP. Given a partial solution, a **completion of it** is obtained by giving values of 0 or 1 to each of the free variables.

The B&B approaches discussed below for this pure 0–1 IP are called **implicit enumeration methods** in the literature. In general the name **implicit enumeration** is used for the class of B&B algorithms designed specifically for the pure 0–1 IP.

### Analysis to be Performed on a Typical CP in Implicit Enumeration

When a new CP is formed after branching, this analysis is applied on it before the lower bounding strategy. The purpose of this analysis is:

- If there is no incumbent yet in the algorithm, it applies simple and efficient tests to check if the system of constraints in the CP is infeasible (i.e., does not have a 0–1 solution). Several such tests have been developed exploiting the special properties of 0–1 variables, we will discuss a few of them for illustrating the main ideas. If one of these tests leads to the infeasibility conclusion, this CP is pruned right away.

The branching constraints may force some free variables to have the same value (0 or 1) in all feasible solutions of the CP. Some of the tests can locate such variables if they exist. If they are identified, they are classified as 0-variables or 1-variables in the CP, making the CP into a smaller problem.

- If there is an incumbent  $\bar{x}$  with objective value  $\bar{z}$  at this stage, we are only interested in feasible solutions of the CP with objective value  $\leq \bar{z}$ . So, in this case we add an additional constraint  $cx \leq \bar{z}$  to the constraints of the CP, and carry out the same tests on this

augmented system. If this augmented system is infeasible, this CP is pruned right away. If some free variables can be shown to have the same value (0 or 1) at all feasible solutions of this augmented system, then the CP is modified by including those variables as 0- or 1-variables.

We will now discuss the analysis to be performed on a typical CP briefly. Consider the CP in which  $\mathbf{U}_0$ ,  $\mathbf{U}_1$ ,  $\mathbf{U}_f$  are the sets of subscripts of the 0-, 1-variables, and the free variables, respectively.  $\mathbf{U}_f = \{1, \dots, n\} \setminus (\mathbf{U}_0 \cup \mathbf{U}_1)$ . Compute the vector  $b' = (b'_i) = b - \sum_{j \in \mathbf{U}_1} A_{.j}$ .

The **fathoming criterion** for this CP is  $b' \geq 0$ . If  $b' \geq 0$ , the completion obtained by giving the value of 0 to all the free variables is optimal to the CP (because of our assumption that  $c \geq 0$ ), and the optimum objective value in it is  $\sum_{j \in \mathbf{U}_1} c_j$ .

If  $b' \not\geq 0$ , several tests are applied to check whether the CP is infeasible (i.e., has no feasible completion) and whether it has a feasible completion better than the current incumbent. Let  $\bar{z}$  be the present upper bound for the minimum objective value in the original problem (i.e., the objective value of the current incumbent), or  $\infty$  if there is no incumbent at this stage. For applying these tests on the CP, the system of constraints to be considered is

$$\sum_{j \in \mathbf{U}_f} a_{ij} x_j \leq b'_i = b_i - \sum_{j \in \mathbf{U}_1} a_{ij}, i = 1 \text{ to } m \quad (8.6.2)$$

$$\sum_{j \in \mathbf{U}_f} a_{m+1,j} x_j \leq b'_{m+1} = \bar{z} - \sum_{j \in \mathbf{U}_1} a_{m+1,j} \quad (8.6.3)$$

$$x_j = 0 \text{ or } 1 \quad \text{for all } j \in \mathbf{U}_f \quad (8.6.4)$$

where for notational convenience we denote  $c_j$  by  $a_{m+1,j}$ . The constraint (8.6.3) is omitted from this system if there is no incumbent at this stage.

**Example 8.6.2:** For an example consider the following original IP.

$$\text{Minimize } z(x) = 6x_1 + 5x_2 + 7x_3 + 4x_4 + 5x_5 + 8x_6$$

$$\begin{aligned}
\text{subject to } 3x_1 - 6x_2 + 2x_3 - x_4 + x_5 + 7x_6 &\leq 12 \\
2x_1 + 3x_2 - 8x_3 - 5x_4 - 3x_5 - 8x_6 &\leq -12 \\
-9x_1 - 7x_2 + x_3 + 2x_4 - 5x_5 - 6x_6 &\leq -11 \\
x_j &\in \{0, 1\} \text{ for all } j = 1 \text{ to } 6.
\end{aligned}$$

Suppose at this stage we have an incumbent  $\bar{x} = (1, 0, 1, 0, 0, 1)^T$  with objective value  $\bar{z} = 26$ .

Consider the CP in which  $U_0 = \{1, 2\}$ ,  $U_1 = \{3\}$ , and  $U_f = \{4, 5, 6\}$ . So in this CP the branching constraints are:  $x_1 = 0$ ,  $x_2 = 0$ , and  $x_3 = 1$ . Fixing these values for these variables, the constraints in the CP in terms of the free variables are:

$$\begin{aligned}
-x_4 + x_5 + 7x_6 &\leq 12 - 2 = 10 \\
-5x_4 - 3x_5 - 8x_6 &\leq -12 - (-8) = -4 \\
2x_4 - 5x_5 - 6x_6 &\leq -11 - 1 = -12 \\
x_4, x_5, x_6 &\text{ are all binary.}
\end{aligned}$$

The objective function in this CP is  $7 + 4x_4 + 5x_5 + 8x_6$ , and since we are only interested in feasible solutions of this CP that are better than the present incumbent, they have to satisfy:  $7 + 4x_4 + 5x_5 + 8x_6 \leq 26$ , or  $4x_4 + 5x_5 + 8x_6 \leq 26 - 7 = 19$ . So, the system (8.6.2) to (8.6.4) corresponding to this CP is:

$$\begin{aligned}
-x_4 + x_5 + 7x_6 &\leq 12 - 2 = 10 \\
-5x_4 - 3x_5 - 8x_6 &\leq -12 - (-8) = -4 \\
2x_4 - 5x_5 - 6x_6 &\leq -11 - 1 = -12 \\
4x_4 + 5x_5 + 8x_6 &\leq 19 \\
x_4, x_5, x_6 &\text{ are all binary. } \bowtie
\end{aligned} \tag{8.6.5}$$

Some of the tests examine each of the constraints in (8.6.2), (8.6.3) individually to check whether it can be satisfied in 0–1 variables. For example, one of the tests is the following.

**Test 1:** In the  $i$ th constraint in (8.6.2), (8.6.3), if  $\sum_{j \in \mathbf{U}_f} (\min\{a_{ij}, 0\}) > b'_i$ , obviously it cannot be satisfied; and hence the system (8.6.2) to (8.6.4) is infeasible and the CP is pruned.

**Example 8.6.3:** As an example, consider the constraint number  $i = 3$  in (8.6.5) which is  $2x_4 - 5x_5 - 6x_6 \leq -12$ . Applying this test on this constraint, the left hand side  $\sum_{j \in U_f} (\min\{a_{ij}, 0\}) = \min\{2, 0\} + \min\{-5, 0\} + \min\{-6, 0\} = 0 - 5 - 6 = -11$ , and the right hand side  $b'_i$  in this constraint is  $-12$ . Since  $-11 > -12$ , this test shows that system (8.6.5) is infeasible (cannot have a 0–1 solution), so this CP should be pruned.  $\bowtie$

The tests may also determine that some of the free variables must have a specific value in  $\{0, 1\}$  for (8.6.2)-(8.6.4) to be feasible. Here is an example of such a test.

**Test 2:** Suppose there is a  $k \in \mathbf{U}_f$  and an  $i$  between 1 to  $m + 1$  such that  $\sum_{j \in \mathbf{U}_f} (\min\{a_{ij}, 0\}) + |a_{ik}| > b'_i$ . Then obviously  $x_k$  must be 0 if  $a_{ik} > 0$ , or 1 if  $a_{ik} < 0$ ; for (8.6.2)-(8.6.4) to be feasible. If such variables are identified by the tests, they are included in the sets of 0- or 1-variables in this CP, accordingly.

**Example 8.6.4:** As a numerical example, suppose the  $i$ th constraint in (8.6.2), (8.6.3) is:

$$6x_{10} - 7x_{11} - 9x_{12} + 9x_{13} \leq -8.$$

Applying the test on this constraint with  $x_k = x_{12}$ , we get the left hand side =  $\min\{6, 0\} + \min\{-7, 0\} + \min\{-9, 0\} + \min\{9, 0\} + |-9| = -7 - 9 + 9 = -7 > -8$ , the right hand side in this constraint. Hence we conclude that  $x_{12}$  must equal 1 (because the coefficient of  $x_{12}$  in this constraint is  $-9$ , negative) in every feasible solution of this CP. Applying the test on the same constraint with  $x_k = x_{13}$ , we conclude that  $x_{13}$  must be 0 in every feasible solution of this CP. Therefore we make  $x_{12}$  a 1-variable, and  $x_{13}$  a 0-variable, take the indices 12, 13 out of the set  $U_f$ , and continue.  $\bowtie$

**Surrogate Constraints:** Let  $\mu = (\mu_1, \dots, \mu_{m+1})$  be a nonnegative vector. Any solution satisfying (8.6.2), (8.6.3) must obviously satisfy

$$\sum_{i=1}^{m+1} \mu_i \left( \sum_{j \in \mathbf{U}_f} a_{ij} x_j \right) \leq \sum_{i=1}^{m+1} \mu_i b'_i \quad (8.6.6)$$

So, if (8.6.6) does not have a 0–1 solution, (8.6.2)-(8.6.4) must be infeasible and the CP can be pruned. A constraint like this obtained by taking a nonnegative linear combination of constraints in the system (8.6.2), (8.6.3), is known as a **surrogate constraint**. For example from the system

$$\begin{aligned} x_1 - x_2 &\leq -1 \\ -x_1 + 2x_2 &\leq -1 \end{aligned}$$

we get the surrogate constraint  $x_2 \leq -2$  by taking the multiplier vector  $\mu = (1, 1)$ , i.e., summing the two constraints. From this surrogate constraint we clearly see that the system has no 0–1 solution, even though we cannot make this conclusion by considering any one of the two original constraints individually. In the same way, often a surrogate constraint enables us to make some conclusions about the system (8.6.2)-(8.6.4), which are not apparent from anyone of the constraints in the system considered individually. If a surrogate constraint has no 0–1 solution, the system (8.6.2)-(8.6.4) is infeasible and the CP is pruned. If some of the free variables must have specific values of 0, 1 in every 0–1 solution for the surrogate constraint, those free variables must have the same specific values in every 0–1 solution for the CP that is better than the current incumbent, and hence they are included accordingly in the sets of 0-, 1-variables defining the CP.  $\bowtie$

We have only discussed a few tests developed in implicit enumeration. There are many others. Software systems for 0–1 IPs based on implicit enumeration use several of these tests to improve the efficiency of the package. For a detailed discussion of useful tests, and methods for generating useful surrogate constraints, see [F. Glover, 1968] and [G. L. Nemhauser and L. A. Wolsey, 1988].

## Lower Bounding Strategies

Let  $\pi = (\pi_1, \dots, \pi_m)$  be a nonnegative vector. Since every feasible solution of (8.6.2) to (8.6.4) is a feasible solution for the problem (8.6.7),  $\sum_{j \in \mathbf{U}_1} c_j +$  (minimum objective value in (8.6.7)) is a lower bound for the minimum objective value in

$$\begin{aligned} & \text{Minimize} && \sum_{j \in \mathbf{U}_f} c_j x_j \\ & \text{subject to} && \sum_{i=1}^m \pi_i \left( \sum_{j \in \mathbf{U}_f} a_{ij} x_j \right) \leq \sum_{i=1}^m \pi_i b'_i \\ & && x_j = 0 \text{ or } 1 \text{ for all } j \in \mathbf{U}_f \end{aligned} \quad (8.6.7)$$

the CP. (8.6.7) is a 0–1 IP with a single constraint, and hence can be solved by algorithms discussed for the knapsack problem. By applying a few steps of the knapsack algorithm on (8.6.7) if we can determine that the lower bound for the minimum objective value in the CP is  $>$  the cost of the present incumbent, then the CP can be pruned. It has been proved that the best  $\pi$ -vector to use for forming the problem (8.6.7) is the negative dual optimum solution associated with the relaxed LP for the CP. See [G. L. Nemhauser and L. A. Wolsey, 1988] for a proof of this result, and other ways of generating and using surrogate constraints effectively.

If all this work determines that (8.6.2)-(8.6.4) is infeasible, the CP is pruned. Otherwise let  $\mathbf{U}'_0, \mathbf{U}'_1$  be the sets of subscripts of the 0- and 1-variables respectively in the CP after augmenting the 0- and 1-variables determined by the tests to  $\mathbf{U}_0, \mathbf{U}_1$  respectively. The set of free variables is  $\mathbf{U}'_f = \{1, \dots, n\} \setminus (\mathbf{U}'_0 \cup \mathbf{U}'_1)$ . Another lower bound for the minimum objective value in the CP is  $\sum_{j \in \mathbf{U}'_1} c_j$ .

### THE ALGORITHM

We now state the algorithm completely. It uses the backtrack search strategy mentioned in Section 8.3.3 with the LIFO selection criterion.

Initially the original problem is the current CP with both the subscript sets of 0- and 1-variables empty. The stack is empty initially.

In a general stage of the algorithm suppose the current CP is defined by the subscript sets  $\mathbf{U}_0, \mathbf{U}_1$  for 0- and 1-variables respectively. Do the following.

Step 1. If the current CP is fathomed update the incumbent, prune the stack and go to Step 2. If the current CP is pruned go to Step 2.

Step 2. If the stack is empty at this stage, the incumbent is an optimum solution of the original problem, terminate. If the stack is empty and there is no incumbent at this stage, the original problem is infeasible, terminate. If the stack is nonempty, retrieve a CP from the stack using the LIFO selection criterion, and make it the new current CP, and go to Step 3.

Step 3. If the current CP is not fathomed apply the tests on it. If the current CP is pruned by the tests, go to Step 2. If it is not pruned by the tests, let  $\mathbf{U}'_0, \mathbf{U}'_1$  be the subscript sets of 0- and 1-variables in the problem after augmenting the new 0- and 1-variables identified by the tests, to  $\mathbf{U}_0, \mathbf{U}_1$  respectively.  $\mathbf{U}'_f = \{1, \dots, n\} \setminus (\mathbf{U}'_0 \cup \mathbf{U}'_1)$  is the subscript set of free variables in the problem.

If  $\mathbf{U}'_f = \emptyset$ , the current CP is fathomed; go to Step 1. If  $\mathbf{U}'_f \neq \emptyset$ , select an  $x_j$  with  $j \in \mathbf{U}'_f$  as the branching variable. Branching generates two candidate subproblems. CP 1 has  $\mathbf{U}'_0, \mathbf{U}'_1 \cup \{j\}$  as the subscript sets for 0- and 1-variables respectively. CP 2 has  $\mathbf{U}'_0 \cup \{j\}, \mathbf{U}'_1$  as the subscript sets for 0- and 1-variables respectively. Add CP 2 to the stack. Make CP 1 the new current CP and continue by applying this Step 3 on it.

For efficient branching variable selection criteria in this algorithm see [G. L. Nemhauser and L. A. Wolsey, 1988].

## 8.7 Advantages and Limitations of the B&B Approach, Recent Developments

In this chapter we discussed the general B&B approach and its application to solve a variety of problems. Various techniques for developing bounding, branching, and search strategies have also been illustrated in these applications. The examples provide insight into how B&B algorithms can be developed for solving integer programs and combinatorial optimization problems.

### Recent Developments, Cutting Planes, Polyhedral Combinatorics, Branch & Cut

Consider the set of feasible solutions,  $\Gamma_I$ , of a pure or mixed integer program. Let  $P_I$  denote the convex hull of  $\Gamma_I$  (i.e., set of all convex combinations of points in  $\Gamma_I$ ). Let  $P$  denote the set of feasible solutions of the LP relaxation of this integer program. Thus  $P_I$  is a subset of  $P$ .

See Figure 8.8 for a 2-dimensional illustration. The dots in the figure (grid points) are the integer points in  $R^2$  ( $x_1, x_2$ - two dimensional plane). The LP relaxation is characterized by 5 linear inequalities in  $x_1, x_2$ ; the half-spaces corresponding to them are shown in solid lines with arrows pointing on the feasible side on each of them, and these are numbered 1 to 5. So,  $P$ , the set of feasible solutions of the LP relaxation, is the region bounded by the solid lines, consisting of both the shaded and dashed regions.

In this integer program, suppose both the variables  $x_1, x_2$  are required to be integral. So,  $\Gamma_I$ , the set of feasible solutions of the integer program, is the set of all grid points inside of  $P$  including those on the boundary of  $P$ , consisting of a total of 25 integer points.

$P_I$ , the convex hull of  $\Gamma_I$ , is the shaded region inside  $P$ . We see from the figure that  $P_I$  can be represented by linear constraints (in  $x_1, x_2$ ) only, without any integer requirements on the variables. However, the linear constraint representation of  $P_I$  requires additional inequality constraints besides those in the linear constraint representation of the LP relaxation  $P$ , these are the ones numbered 6, 7, 8, 9 in the figure with dashed lines with arrows on each of them pointing on the feasible side

of each of them.

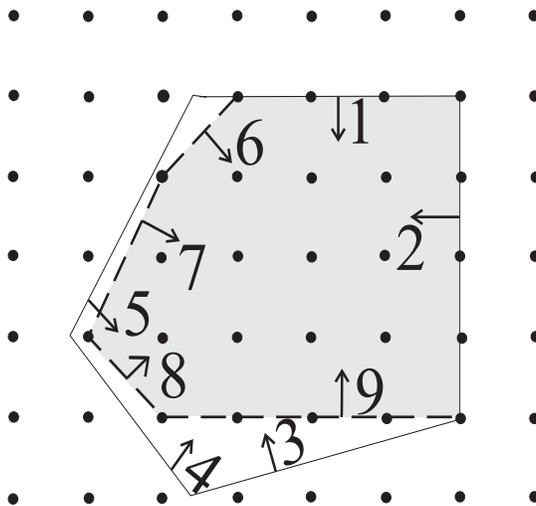


Figure 8.8:  $P, \Gamma_I, P_I$  corresponding to an integer program in two variables  $x_1, x_2$ . See above para for details. Constraints 1 to 5 are constraints in the linear relaxation. Constraints 6, 7, 8, 9 are cuts to augment to the LP relaxation, to characterize the convex hull of integer feasible solutions.

This feature that  $P_I$  can be characterized through a system of linear constraints (with no integer restrictions on the variables), is true for integer and mixed integer programs in any number of variables. However, this requires other constraints in addition to those in the LP relaxation. These additional constraints in the linear constraint representation of  $P_I$  are called **cuts** or **cutting planes** or **valid cuts** or **valid inequalities**.

Augmenting the system of inequalities of the LP relaxation with all the necessary cuts to characterize  $P_I$  satisfies two important properties:

1. It keeps all the points in  $\Gamma_I$  feasible (i.e., every feasible solution of the original integer program is feasible to the augmented system).
2. It removes regions in  $P$  not contained in  $P_I$ .

Once a complete linear constraint representation of  $P_I$  is available, the original integer program is the same as minimizing the original objective function subject to all these constraints to obtain an extreme point optimum. Hence it can be solved as a linear program with no integer restrictions on the variables.

The branch of mathematics dealing with the problem of obtaining all the necessary cuts for integer programs is called **cutting plane theory**. Unfortunately, in integer programs with more than three variables, the number of cuts needed to characterize  $P_I$  is typically very large (tends to grow exponentially with the number of integer variables), so the general cutting plane theory is of very limited practical use.

But it has been found that by adding just a selected few good cuts to the LP relaxation, one can get very high quality lower bounds for the integer program. The branch of mathematics dealing with identifying good families of cuts (linear constraints) to add to the LP relaxation of an integer program to get high quality lower bounds for the integer program's minimum objective value is called **polyhedral combinatorics**. This is a deep and very exciting area of research in integer programming. The B&B algorithm based on lower bounds computed using these additional cuts is known as the **branch and cut (B&C) algorithm**. Branch and cut algorithms are always specialized algorithms for a specific type of integer program, because identifying good families of cuts to use in it requires deep knowledge about the geometric structure of the original integer program.

On some problems with nice mathematical structure (such as the TSP, the knapsack problem, and certain special types of pure 0–1 IPs) great strides have been made recently in developing B&B, and B&C algorithms for solving large scale instances of the problem by exploiting their special structure.

Outside of this class of problems with nice mathematical structure,

the performance of B&B algorithms is uneven, particularly as the size of the instance (as measured by say, the number of 0–1 variables in the model) becomes large. On such problems the B&B algorithm may require an enormous amount of computer time, as the number of nodes examined in the search tree grows exponentially with the size of the instance. However, it usually produces very good incumbents early in the search effort. Even though these early incumbents are not guaranteed to be optimal to the problem, they usually turn out to be very close to the optimum. This is what makes the B&B approach useful in applications.

## 8.8 Exercises

**8.8.1:** Solve the 0–1 knapsack problem with 10 available objects with the following data, to maximize the value loaded into a knapsack of weight capacity 40 weight units.

| Object | Weight | Value |
|--------|--------|-------|
| 1      | 19     | 380   |
| 2      | 15     | 225   |
| 3      | 20     | 320   |
| 4      | 8      | 96    |
| 5      | 5      | 70    |
| 6      | 7      | 126   |
| 7      | 3      | 30    |
| 8      | 2      | 22    |
| 9      | 4      | 68    |
| 10     | 42     | 900   |

**8.8.2:** Consider an undirected network consisting of nodes (represented by little circles with its number entered inside, in a figure of the network); and edges, each of which is a line joining a pair of distinct nodes. A **clique** in such a network is a subset of nodes  $\mathbf{N}$  satisfying the property that every pair of nodes in  $\mathbf{N}$  is joined by an edge in the network.

As an example, in the network in Figure 8.9 with 11 nodes; the

subset of nodes  $\{1, 10, 9, 2\}$  is not a clique because nodes 1 and 9 in this subset are not joined by an edge in the network. But the subset of nodes  $\{1, 2, 3, 5\}$  is a clique because every pair of nodes in this subset is joined by an edge in the network.

The cost of including each node in a clique is given. Typically, these cost coefficients are negative. The cost of a clique is defined to be the sum of the cost coefficients of nodes in it. For example, the cost of the clique  $\{1, 2, 3, 5\}$  in the network in Figure 8.9 is  $-2 - 5 - 7 - 1 = -15$ .

Develop a B&B algorithm for finding a minimum cost clique. And find a minimum cost clique in the network in Figure 8.9 using your algorithm.

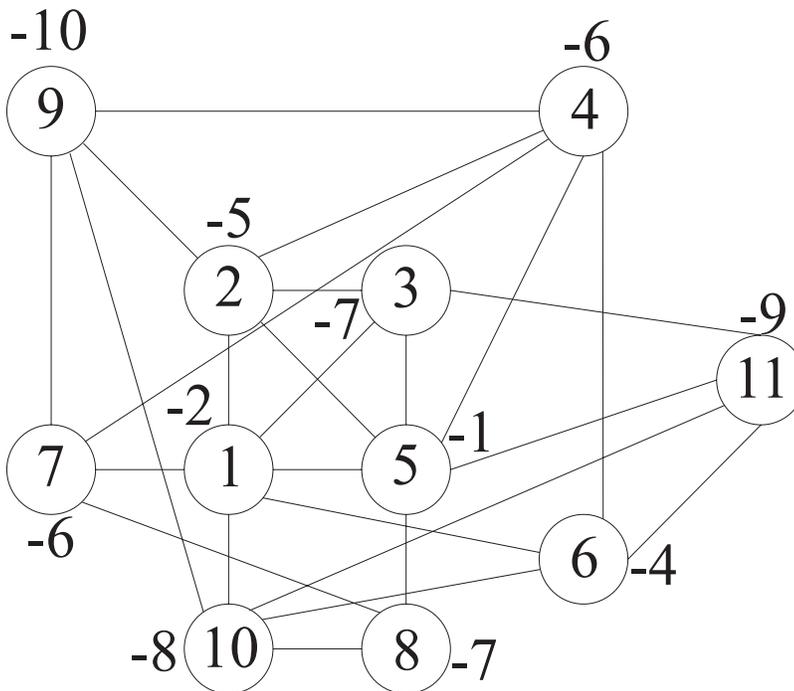


Figure 8.9: The negative number by the side of each node is its cost coefficient.

**8.8.3:** Formulate and solve the following multiconstraint 0–1 knap-

sack problem. The total value included in the knapsack is to be maximized subject to the knapsack's weight and volume constraints.

| Object                 | Weight (lbs.) | Volume (ft <sup>3</sup> ) | Value (\$) |
|------------------------|---------------|---------------------------|------------|
| 1                      | 20            | 41                        | 84         |
| 2                      | 12            | 51                        | 34         |
| 3                      | 7             | 24                        | 31         |
| 4                      | 75            | 40                        | 14         |
| 5                      | 93            | 84                        | 67         |
| 6                      | 21            | 70                        | 65         |
| 7                      | 75            | 34                        | 86         |
| 8                      | 67            | 41                        | 98         |
| 9                      | 34            | 49                        | 50         |
| 10                     | 28            | 27                        | 7          |
| Knapsack's<br>capacity | 190           | 250                       |            |

([W. Shih, April 1979])

**8.8.4:** Solve the following MIPs by the B&B approach

$$\begin{aligned}
 &\text{Maximize} && 2x_1 + x_2 + 3y_1 + 4y_2 \\
 &\text{subject to} && x_1 + 3x_2 - y_1 + 2y_2 \leq 16 \\
 &&& -x_1 + 2x_2 + y_1 + y_2 \leq 4 \\
 &&& x_1, \quad x_2, \quad y_1, \quad y_2 \geq 0 \\
 &&& x_1, \quad x_2 \text{ are integer}
 \end{aligned}$$

$$\begin{aligned}
 &\text{Maximize} && 4y_1 + 5x_1 + x_2 \\
 &\text{subject to} && 3y_1 + 2x_1 \leq 10 \\
 &&& y_1 + 4x_1 \leq 11 \\
 &&& 3y_1 + 3x_1 + x_2 \leq 13 \\
 &&& y_1, \quad x_1, \quad x_2 \geq 0
 \end{aligned}$$

$x_1, x_2$  are integer

## 8.9 References

O. H. IBARRA and C. E. KIM, October 1975, "Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems", *Journal of the ACM*, 22, no. 4 (463-468).

F. GLOVER, 1968, "Surrogate Constraints", *Operations Research*, 16, no. 4, (741-749).

A. H. LAND, and A. G. DOIG, 1960, "An Automatic Method for Solving Discrete Programming Problems", *Econometrika*, 28 (497-520).

K. G. MURTY, C. KAREL, and J. D. C. LITTLE, 1962, "The Traveling Salesman Problem: Solution by a Method of Ranking Assignments", Case Institute of Technology. Copy can be seen in "Selected Publications" at website: <http://www-personal.engin.umich.edu/~murty/>

G. L. NEMHAUSER, and L. A. WOLSEY, 1988, *Integer and Combinatorial Optimization*, Wiley, NY.

W. SHIH, April 1979, "A Branch and Bound Method for the Multiconstraint Zero-one Knapsack Problem", *Journal of the Operational Research Society*, 30, no. 4, 369-378.

# Index

For each index entry we provide the section number where it is defined or discussed first.

## **B& B 8.2**

Backtrack search 8.3.3

Branch and bound 8.1

Branching 8.2

    Operation 8.2, 8.3

    Strategy 8.3

    Variable 8.3

## **C P 8.2**

Candidate problem 8.2

Cutting plane theory 8.7

## **Fathomed 8.2**

Fathoming strategy 8.4

Free variable 8.4

## **General MIP 8.5**

    B& B for 8.5

Greedy heuristic 8.4

## **Implicit enumeration 8.6**

### **List 8.3.3**

Live node 8.3.3

Lower bounding 8.2, 8.3.1

Lower bounds 8.2

    Quality of 8.2

## **Partial enumeration 8.2**

Polyhedral combinatorics 8.7

Pruning 8.2

Pure 0-1 IP 8.6

    B&B for 8.5

## **Relaxed optimum 8.3.1**

Relaxed problem 8.3.1

## **Search 8.3.3**

    Strategy 8.3.3

    Tree 8.3.3

Stack 8.3.3

## **Tests 8.6**

## **Valid cuts 8.7**

Valid inequalities 8.7

## **0-1 Knapsack problem 8.4**

    LP relaxation of 8.4