Chapter 6

MIXED INTEGER LINEAR PROGRAMMING

Introduction

There are linear programming problems that require integer values for some or all the decision variables. For example, integer quantities are necessary for activities associated with machines, vehicles or people. These problems with some of the variables having integer values are known as Mixed Integer Linear Programming (MILP) problems. The use of integer variables makes possible the formulation of models of many problems for which only an approximation was available previously. Industrial applications of mixed integer programming include: flowsheeting optimization, optimal scheduling of batch plants, heat and mass exchanger networks, multiphase chemical equilibrium, blending in limited tanks, optimal feed location in distillation and reaction path synthesis. Others include capital budgeting, most valuable mix and equipment scheduling.

In this chapter, the mathematical representation of a mixed integer linear programming is given to describe the mathematical structure of such problems. This is followed by an algorithm to solve problems involving MILP, and its use is illustrated by solving a simple problem. A few examples illustrating special cases of MILP are given also and explained in detail. Also, standard computer codes are described for solving large MILP's. Finally, the important application of optimal scheduling for a multi-product batch plant will be given, and this will include converting this scheduling problem into a mixed integer mathematical model that is then solved using GAMS, the General Algebraic Modeling System for optimization. The computer codes needed for representing the problem as well as the output solution are detailed.

General Statement of Mixed Integer Linear Programming (MILP)

MILP problems require maximizing or minimizing a function subject to linear equality or inequality constraints with integer restrictions on some or all the variables. The mathematical statement of mixed integer linear programming can be expressed as:

$$\text{(MILP)}\ \max\ \{cx + hy\text{: } \boldsymbol{A} + \boldsymbol{G} \leq b,\ x \in \boldsymbol{R}_p^+, y \in \boldsymbol{Z}_n^+, \} \qquad (6\text{-}1)$$

where $\boldsymbol{Zn}$ is a set of n dimensional vector of positive integers and $\boldsymbol{Rp}$ is a set of p-dimensional positive real vectors. The variables or unknowns are $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_p)$. $\boldsymbol{A}$ and $\boldsymbol{G}$ are $m \times n$ and $m \times p$ matrices respectively. The objective function is $z = cx + hy$ with c and h being n and p ordered vectors respectively [1].

The MILP has two special cases: Linear Programming (LP) that has all continuous variables and Integer Programming (IP) that has only integer variables. The mathematical statement of an integer linear programming problem is the same as the linear programming model, but with an additional restriction that the variables must take on integer values. It is expressed in the following form in summation notation:

$$\text{minimize}: \ z = \sum_{j=1}^{p} h_j y_j \tag{6-2a}$$

$$\text{subject to:} \ \sum_{j=1}^{p} g_{ij} y_j \leq b_i \ \text{ for } i = 1,2, ..., m \tag{6-2b}$$

$$y_j \geq 0, \text{ for } j = 1, 2, ..., p$$

$$y_j \ \text{ integer, for } j = 1, 2, ..., p$$

In matrix notation, after Equation 6-1, Equations 6-2a and 6-2b for IP are expressed as:

$$\text{(IP) max } \{hy\text{: } \boldsymbol{G} \leq b, y \in \boldsymbol{Z}_n^{+}\} \tag{6-3}$$

The mathematical statement of linear programming problem after Equation (1) is:

$$\text{(LP) max } \{cx\text{: } \boldsymbol{A} \leq b, x \in \boldsymbol{R}_{\text{p}}^{+}\} \tag{6-4}$$

As mentioned above, LP is considered to be a special case of MILP in which the variables have no integer restrictions and can assume any positive real value. The deletion of the integer restriction in a mixed integer problem reduces it to an ordinary linear program in which all the variables are continuous, and this is used in algorithms to solve MILP's.

Integer programming (IP) has a special case that is used in applications involving a number of interrelated "yes-or-no decisions". When integer problems are restricted to values of zero and one, this is the special case of general integer programming called Binary Integer Programming (BIP). An example of such a model is the capital budgeting problem in which n projects are competing for limited resources such as equipment, manpower and money. The objective is to schedule projects to yield the largest profit while satisfying the specified limitations. Here, y_j can be defined as a binary variable representing the *j*-th project so that $y_j = 1$ (or 0) if the *j*-th project is scheduled (not scheduled). This problem is described by Ecker and Kupferschmidt [2] and Ravindran, et.al. [3].

Another example is the knapsack problem where the most valuable mix is determined from among n items to be packed in a knapsack, providing that the total amount of volume of the selected items does not exceed the capacity of the knapsack. Here too, $y_j = 1$ or 0 depending on whether or not item j is selected. This problem is described by Ecker and Kupferschmidt [2].

The mathematical statement of a binary integer-programming problem is the same as an integer programming statement with the additional restriction that all the variables are binary variables. It is expressed in the following form in the summation notation:

$$\text{minimize}: \quad z = \sum_{j=1}^{n} h_j y_j$$

$$\text{subject to:} \quad \sum_{j=1}^{p} g_{ij} y_j \leq b_i \quad \text{for } i = 1,2, ..., m \qquad (6\text{-}5)$$

$$y_j = 0 \text{ or } 1, \text{ for } j = 1,2, ..., p$$

Perspective on Solving Integer Programming Problems

The two primary determinants of computational difficulty for an IP problem are the number of integer variables and the structure of the problem. This situation is in contrast to linear programming, where the number of (functional) constraints is much more important than the number of variables. In integer programming, the importance of constraints is secondary to the other two factors. For MILP problems, it is the number of integer variables that is important, because the computational time increases tremendously as the number of integer variable increases.

Integer programming problems frequently have some special structure that can be exploited to simplify and solve very large problems successfully. Special purpose algorithms designed specifically to exploit certain kinds of special structures are becoming increasingly important in integer programming.

There are three generally used methods for solving integer-programming problems: LP-relaxation, cutting plane and branch and bound. The first one is a simple approximate method and the third one is considered the best of the three. In LP-relaxation the linear programming problem is solved ignoring the integer restriction, and then the noninteger values in the resulting solution are rounded-off to integer values. Sometimes, sequences of LP-relaxations for portions of an IP problem are used to solve the overall IP problem effectively. Although this is often adequate, this approach is not always accurate. One of the drawbacks is that the optimal linear programming solution may not necessarily remain feasible after it is rounded-off. Even if the optimal linear programming solution is rounded off successfully, there is no guarantee that this rounded-off solution will be the optimal integer solution. Moreover, for large problems, such a procedure can become computationally expensive. For example, if the optimal LP solution is $x_1 = 3.2$ and $x_2 = 4.6$, then there are four different combinations of integer values to x_1 and x_2 that are close to their continuous values. (3, 4), (3, 5), (4, 4), and (4, 5). If the feasible solutions are selected from these four, then the one that gives the smallest value of the objective function (if minimizing) will be an approximate integer solution. For 10 integer variables, this gives $2^{10} = 1024$ combinations of integer solutions that will have to be evaluated according to Ravindran, et.al. [3]. Then, after performing these evaluations, there is no guarantee that an optimal integer solution has been found.

The cutting plane algorithm solves a sequence of successively tighter LP relaxation problems, hoping to produce an optimal integer solution. Details are given by Nemhauser, et.al.

[1]. The algorithm eliminates parts of the feasible region that do not contain feasible integer solutions. However, it is not unusual to have a very large number of cuts required for convergence.

The most widely used method for solving both integer and mixed integer programming problems is the branch-and-bound algorithm. Most commercial computer codes for solving integer-programming problems use this approach [3,4]. The method performs an efficient enumeration of a small fraction of the possible feasible integer solutions to locate the optimum. In the next section, the branch and bound technique is described in detail for IP, and it is extended for the important special case of BIP and the more general case of MILP.

The Branch and Bound Technique

A bounded integer-programming problem has a finite number of feasible solutions, and it is natural to consider using an enumeration procedure for finding an optimal solution. Unfortunately, this finite number can be, and usually is, very large; and exhaustive enumeration has been found to be prohibitively time consuming for such problems [2]. Therefore, it is imperative that an enumeration procedure be structured so that only a small fraction of the feasible solutions are examined.

The basic idea of the branch-and-bound technique is to divide and conquer. If the original problem is very large, then it would be difficult to solve it directly; and hence it is divided into smaller and smaller subproblems until these subproblems can be solved easily or conquered. To divide (branch) the original problem into smaller subproblems, the entire set of feasible solutions is partitioned into smaller and smaller subsets; and for each one, an upper bound for the value of the objective function is obtained from the solutions within that subset (when maximizing). The conquering (fathoming) is done in two parts. Firstly, the bounds for the best solution in the subset are found; and then the subset is discarded if its bound indicate that it cannot possibly contain an optimal solution for the original problem [5]. The subset with the highest upper bound is partitioned further into subsets. Their upper bounds are obtained in turn and used as before to exclude some of these subsets from further consideration. From all the remaining subsets, another one is selected for further partitioning and so forth. This process is repeated until a feasible solution is located such that the corresponding value of the objective function is greater than the upper bound for any of the other subsets. Such a feasible solution must be optimal since none of the subsets can contain a better solution.

A Branch-and-Bound Algorithm for General Integer Programs

This algorithm has four stages as described below after Ecker and Kupferschmid [2]. First, the original problem is solved with LP relaxation (Step 0. Initialize). Then the first step involves partitioning the original set (problem) into two subsets (subproblems) by adding additional constraints (Step 1. Branch). Objective function values from the newly partitioned subsets are obtained in the second step by solving the LP's for the subsets (Step 2. Bound). In the third step, all of the subsets that cannot contain the optimal solution are designated for no further evaluation. This is called fathoming (Step 3. Fathom). The fourth step tests if there are any more subsets to be fathomed; and if there are, the algorithm is invoked again (Step 4. Test). These steps are described in detail as follows for maximizing the objective function of the integer programming problem:

Step 0. Initialize. (Locate upper and lower bounds)

Solve the original problem by linear programming relaxation. If all the constraints are satisfied by the solution, then the optimal integer solution for the problem has been found.

The linear programming relaxation solution provides an upper bound, z_U, to the problem because the optimal integer solution cannot have an objective function value larger than the linear programming relaxation solution. The imposition of integer restriction on $\boldsymbol{x}$ can only make the solution worse.

If an integer solution has not been found, then a lower bound z_L for the optimal objective function value is found that is equal to the objective value at some point that is feasible for the integer program. This could be where all of the variables are zero or some comparable solution that satisfies all the constraints and that will surely be smaller than the final optimal value.

If no such feasible point is readily known, set $z_L = -\infty$. This lower bound solution is also designated as the incumbent solution. This means that it is the best integer solution obtained so far. When a better integer feasible point is obtained as the solution proceeds, then that would be the new incumbent solution.

Step 1. Branch. (Partition problem into two subsets)

Select a noninteger basis variable from the LP solution to the problem (initially, the LP relaxation solution) and partition the set into two subsets. A subset is obtained from a set by introducing an additional constraint to the set (branching). The additional constraint depends on the noninteger basis variable that is selected for branching.

If there are more than one noninteger basis variables in the solution, then any one of them can be selected for branching, and the solution may move more rapidly by selecting the variable with the largest fractional value [3].

Thus, branching is accomplished by adding constraints to the LP problem to exclude the noninteger values of the chosen basis variable. For example, if the output solution has the values $x = [0, 2.5, 3]$, then this set is partitioned further into two subsets by adding an additional constraint to exclude the noninteger value of the variable. (In this case, x_2). The additional constraints for the two subsets would be $x_2 \leq 2$ and $x_2 \geq 3$ respectively.

Step 2. Bound. (Solve LP's from subsets)

Solve the two new linear programs that are obtained by appending the extra constraint as a result of Step 1, to the original programming relaxation. These are designated as subsets, and their resulting optimal values (if they are not infeasible) would be the upper bound z_U for that branch when the subset is developed because additional integer constraints are added in expanding branches.

Step 3. Fathom. (Test of the LP objective function values for the subsets to determine if no further evaluation)

Examine the subsets that contain the optimal points, and fathom a subset if:

(a) $z_U \leq z_L$, i.e. subset objective function value is less than the lower bound, and no further evaluations are needed.

(b) The subset has no feasible points, and no further evaluations are needed.

(c) If z_U is an integer feasible solution and $z_U > z_L$, then this is the new incumbent solution, since it is the best integer solution obtained thus far.

Step 4. *Test.* (Determine remaining subsets to be evaluated)

Select a subset among those from Step 1 that has noninteger values for branching. If all subsets have been fathomed, the incumbent solution is optimal for IP. Otherwise, return to Step 1.

If the objective is to minimize rather than maximize the objective function, the procedure is unchanged except that the roles of the upper and lower bounds are reversed. Thus z_L would be replaced by z_U and vice versa, ∞ becomes $-\infty$, and the directions of the inequalities in the branch and bound algorithm would be reversed.

To apply the branch and bound algorithm, rules are needed to determine the selection of variables for branching and the order to follow the branches along with determining the lower bounds on the objective function value. The two most popular branch rules are the *best-bound rule* and the *newest bound rule*.

The best-bound rule selects the subset having the most favorable bound (the largest upper bound in the case of maximization) because this subset would seem to be the most promising one to contain an optimal solution.

The newest bound rule selects the most recently created subset that has not been fathomed for further branching. A tie between subsets created at the same time is broken by taking the one with the most favorable bound.

Also, branches can be developed by the breadth first rule and the depth first rule. The breadth first rule has the subsets generated at the current depth of the branching evaluated before moving further down. The depth first rule has the subsets generated in the center, expanded down as far as possible before evaluating subsets on the left or right. In the next section this method is illustrated by solving a simple integer programming problem and a binary integer-programming problem.

A Branch-and-Bound Example for Integer Programming

The branch-and-bound algorithm is illustrated in solving the following integer programming example problem after Ecker and Kupferschmid [2].

$$\begin{aligned} \text{IP: maximize: } z(x) = \; & -3x_1 + 7x_2 + 12x_3 && \text{(6-6a)} \\ \text{subject to: } \; & -3x_1 + 6x_2 + 8x_3 \le 12 \\ & 6x_1 - 3x_2 + 7x_3 \le 8 \\ & -6x_1 + 3x_2 + 3x_3 \le 5 \\ & x_1, x_2, x_3, \text{ nonnegative integers} && \text{(6-6b)} \end{aligned}$$

The above can be represented as {maximize $z(x)$, subject to $\boldsymbol{A}_x \le b$, $x \in \boldsymbol{F}$} where $\boldsymbol{F}$ is the set of all nonnegative vectors $x \in \boldsymbol{R}_3$ such that all three linear inequality constraints are satisfied. The following gives the steps in solving this problem by the branch and bound algorithm that was described previously.

Step 0 Initialize.

The simplex method is used to solve the linear programming problem without the requirement that the x_j's be integer. This is called linear programming relaxation and is designated LP-1. The result is:

$$\boldsymbol{x} = [0, 0.30, 1.3]^T, \; z = 17.4$$

This solution has noninteger components, and thus it is not optimal for IP. However, the optimal integer solution can not have an objective function larger than 17.4, since the imposition of integer restrictions on $\boldsymbol{x}$ can only make the LP solution worse, i.e. the optimal solution can not be improved by adding constraints. Thus, the upper bound, z_U, for this set is 17.4.

To establish a lower bound on the objective function value we note that $\boldsymbol{x} = 0$ is feasible for IP and yields an objective value of $z(\boldsymbol{x}) = 0$. Thus, the maximum value of IP is surely larger than $z_L = 0$, because we can do that well by selecting $\boldsymbol{x} = 0$.

We could use $z_L = -\infty$ instead, and the algorithm would still work. However, the way z_L is employed in the bounding step, it is sometimes faster and convenient to start with a tighter lower bound of $z_L = 0$. We use this value and declare $\boldsymbol{x} = [0, 0, 0]^T$ to be the incumbent solution which means that $\boldsymbol{x} = [0, 0, 0]^T$ is the best feasible solution obtained.

As we proceed, the incumbent solution is reset to any feasible solution that has a better (greater in case of maximization) value than the previous incumbent solution (if any); and at the end of the procedure, the current incumbent solution is declared to be the optimal value for the original problem.

Step 1. Branch.

According to the algorithm statement, either x_2 or x_3 can be chosen as the variable on which to branch, and the algorithm gives procedures for this selection. Using x_2, LP-1 is partitioned into two linear programs having additional constraints $x_2 \leq 0$ and $x_2 \geq 1$, because x_2 must be integer. The optimal solution to IP must be in:

$$\text{either } \boldsymbol{F} \cap \{x \mid x_2 \leq 0\} \qquad \text{or} \qquad \boldsymbol{F} \cap \{x \mid x_2 \geq 1\}$$

The variable x_2 is constrained to be nonnegative, and every point in the left-hand subset has $x_2 = 0$. This creates the following two new linear programming problems that are solved in step 2.

$$\begin{array}{llll} \text{LP-2 max:} & -3x_1 + 7x_2 + 12x_3 & \text{LP-3 max:} & -3x_1 + 7x_2 + 12x_3 \end{array} \qquad \text{(6-7a)}$$

$$\begin{array}{llll} \text{subject to:} & -3x_1 + 6x_2 + 8x_3 \leq 12 & \text{subject to:} & -3x_1 + 6x_2 + 8x_3 \leq 12 \\ & 6x_1 - 3x_2 + 7x_3 \leq 8 & & 6x_1 - 3x_2 + 7x_3 \leq 8 \\ & -6x_1 + 3x_2 + 3x_3 \leq 5 & & -6x_1 + 3x_2 + 3x_3 \leq 5 \\ & x_2 \leq 0 \text{ new constraint} & & x_2 \geq 1 \text{ new constraint} \end{array} \qquad \text{(6-7b)}$$

Step 2. *Bound.*

The two linear programming problems obtained by adding the extra constraints to the original linear programming relaxation are solved, and the results are given below. These solutions establish a new upper bound on the IP objective function from each of the subsets produced by the branch.

$$\begin{array}{ll} \text{maximize } z(x) & \text{maximize } z(x) \\ x \in \boldsymbol{F} & x \in \boldsymbol{F} \\ \boldsymbol{A}_x \leq b & \boldsymbol{A}_x \leq b \\ \text{subject to } x_2 = 0 & \text{subject to } x_2 \geq 1 \\ & \\ x = [0, 0, 1.1] & x = [0.7, 1, 1] \\ z = 13.7 & z = 17 \end{array}$$

Step 3. *Fathom.*

A subset requires no further evaluation (fathomed) if it satisfies any of the three conditions given in step 3 of the algorithm. Checking the node conditions in step 2, neither of the solution contains integer optimal solutions, and both preceding subsets must be included in further consideration:

13.7 is greater than $z_L = 0$	
17 is greater than $z_L = 0$	$\Rightarrow$ cannot fathom by (a)
neither subproblem is infeasible	$\Rightarrow$ cannot fathom by (b)
neither subproblem has an integer solution that is greater than $z_L = 0$	$\Rightarrow$ cannot fathom by (c)

Step 4. *Test.*

Both subsets remain unfathomed, so step 1 of the algorithm is repeated. The iteration continues until no subsets remain to be fathomed.

An iteration through the algorithm is one application of steps 1 through 4, and many such iterations may be performed before the optimal solution is found. The result of the first iteration is shown in Figure 6-1 in a branching diagram. It is often convenient and helpful to keep track of the solution process by drawing such a diagram. Here, the subproblems are drawn as nodes of a binary tree, and they are connected by links that show how the branching was performed. For this reason, the subproblems are also referred to as nodes.

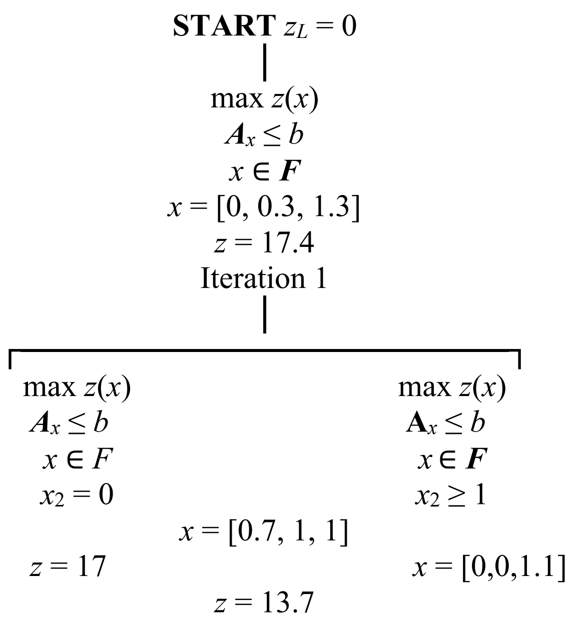


Figure 6-1 Branching Diagram through Iteration 1

From Figure 6-1, we see that the first iteration of the problem yields two unfathomed nodes. Since both nodes have to be fathomed, one of the nodes is chosen to begin the second iteration. Selecting the left node, the subproblem has the LP solution $x = [0, 0, 1.1]$, and so x_3 is chosen as the variable on which to branch.

Two additional constraints are introduced to exclude a noninteger value of x_3 i.e., $x_3 \geq 2$ and $x_3 \leq 1$. These inequalities are used to form the new left and right subproblems as shown at the bottom of Figure 6-2. One of the two new nodes in Figure 9-2 is fathomed because it is infeasible. As mentioned before, a node is fathomed for an infeasible subproblem because it means that there are no points that satisfy both the original constraints and those added in branching. As the constraint set is empty, it cannot contain the optimal point for IP. This subset of F is therefore excluded from further consideration by condition (b) of the step 3 in the algorithm statement, and the same has been noted in the branching diagram.

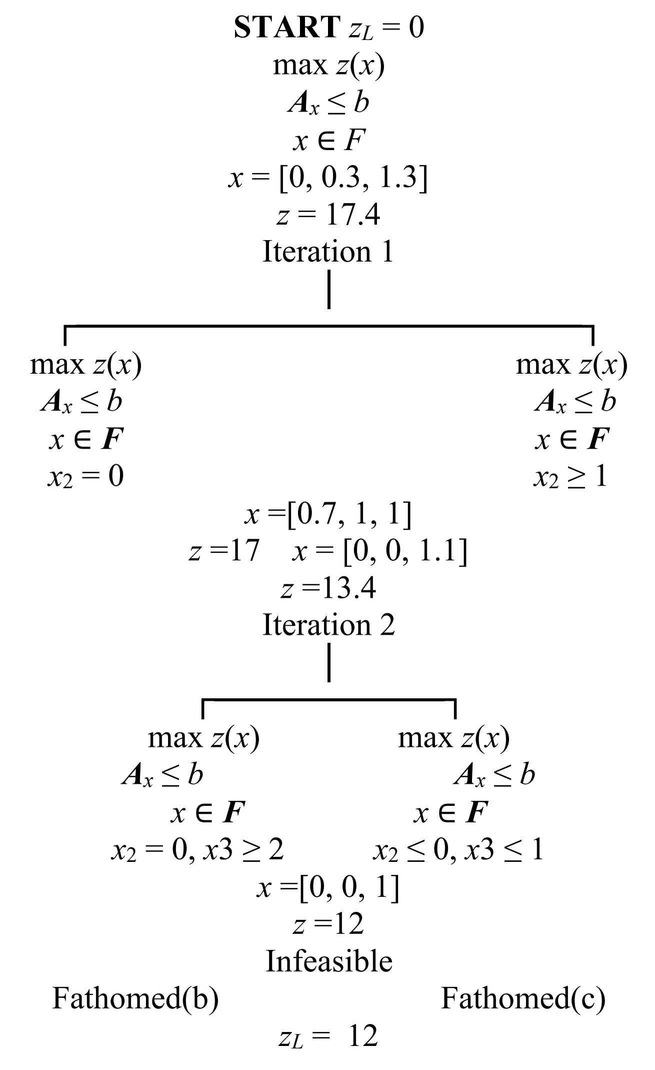


Figure 6-2 Branching Diagram through Iteration 2

The solution for other new subproblem at iteration 2 (Figure 6-2) is an integer solution, $\boldsymbol{x}$ = [0, 0, 1]. Also, $z = 12 > z_L = 0$, so it is fathomed by condition (c). The maximum value for this branch is obtained at the integer solution [0, 0, 1], which means that there are no integer solutions in on this branch having an objective value higher than $z = 12$. Thus, it is not necessary to consider this branch further. If it turns out that this subset contains the optimal point for the integer program, then it must be the point $x = [0, 0, 1]$.

An integer point has been found with an objective function value of more than the current lower bound of $z_L = 0$. The existing lower bound is updated to $z_L = 12$ and $x = [0, 0, 1]$ is declared to be the new incumbent solution. At this stage, one can be sure that the maximum value of the integer program cannot be smaller than $z = 12$ because $x = [0, 0, 1]$ is feasible and it yields an integer objective function value of $z = 12$.

Now that the lower bound value has been changed, it is necessary to evaluate the nodes from the other branch. When the remaining nodes are fathomed, the present incumbent solution is compared to these results. Since the other node has $z = 17 > z_L = 12$, branching is continued.

In Figure 6-3 the third iteration is shown that begins with a branching on x_1 from the unfathomed node on the right. This yields two nodes, and one is an infeasible subproblem without a solution that satisfies the original problem with the additional constraints. This node is fathomed by condition (b). The other node is feasible with a non-integer optimal point, and it cannot be fathomed because it has $z = 16.8 > z_L = 12$, so further branching is required.

The remaining sub problem solution has two variables, x_2 and x_3, with noninteger values, and so branching can be done on either of those variables. Selecting x_3, because it has the largest fractional value, the solution process is continued; and the results are shown in Figure 6-4.

In iteration 4, the right subproblem is fathomed, as it is infeasible. The other new subproblem having a noninteger solution cannot be fathomed because it has $z = 15.6 > z_L = 12$. Another iteration is required, using either x_1 or x_2. Selecting x_2, the procedure is continued, and the results are shown in Figure 6-5 for the entire problem.

In iteration 5, the solution of these two subproblems shows that one of them is infeasible and the other has $z = 15$ at the integer point $x = [2, 3, 0]$. Therefore, both nodes are fathomed, and no further branching is required. Also, $z = 15 > z_L = 12$, so the lower bound is updated to $z_L = 15$ and $x = [2, 3, 0]$ is the new incumbent solution. Applying the convergence test of the algorithm (step 4), the algorithm stops because no unfathomed subsets remain. Therefore, the incumbent solution $x = [2, 3, 0]$ with $z = 15$ is declared to be the optimal solution to the integer problem.

In this example, the optimal point was obtained from the solution of the last subproblem generated in the final iteration. However, this is not always the case, and many times the optimal point is found prior to the final iteration. However, all nodes have to be fathomed to locate the global optimum among the local optima.

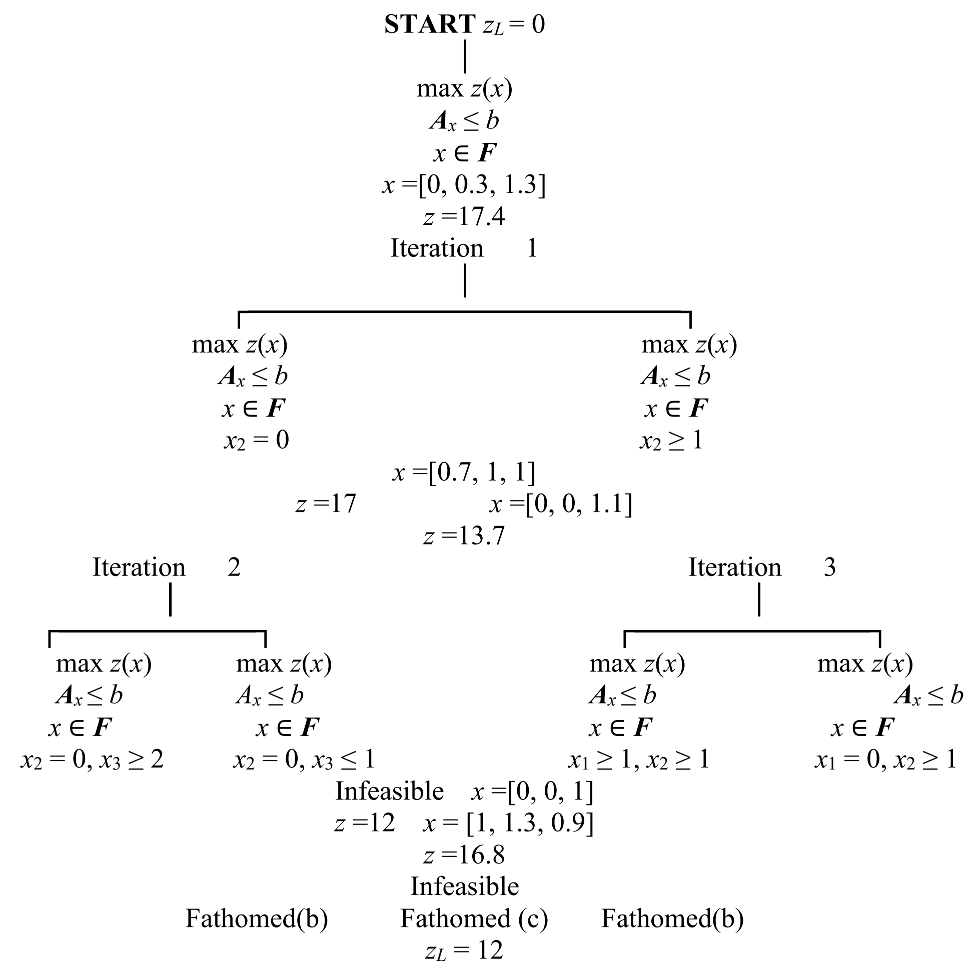


Figure 6-3 Branching Diagram through Iteration 3

The Order of Selecting Unfathomed Nodes

Had the problem been solved depth first with the right-hand branch, it would not be necessary to expand the branch on the left because the integer feasible value of $z = 15$ is greater than $z = 13.7$ for the left branch. See Figure 6-6. It is quite difficult to tell in advance which subset strategy will work best for a particular problem. However, sometimes an intelligent guess can be made on which strategy to select. For example, in this problem the right subproblem generated at Iteration 1 had a higher optimal value than the left subproblem, 16.8 vs. 12. It would be reasonable to expect that following the right-hand branch might yield an integer point with an objective value high enough to fathom the left node. Computer programs incorporate heuristics to assist in making decisions about ways to order the branching.

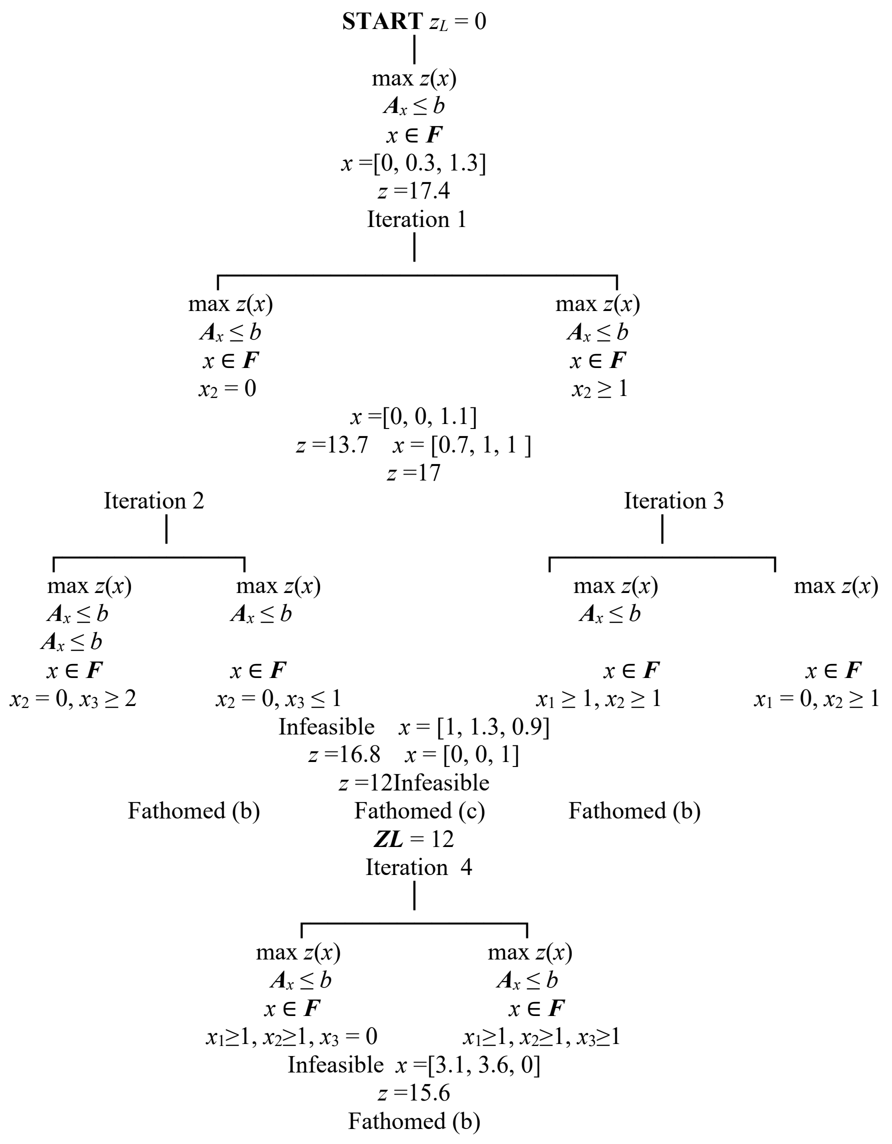


Figure 6.4 Branching Diagram through Iteration 4

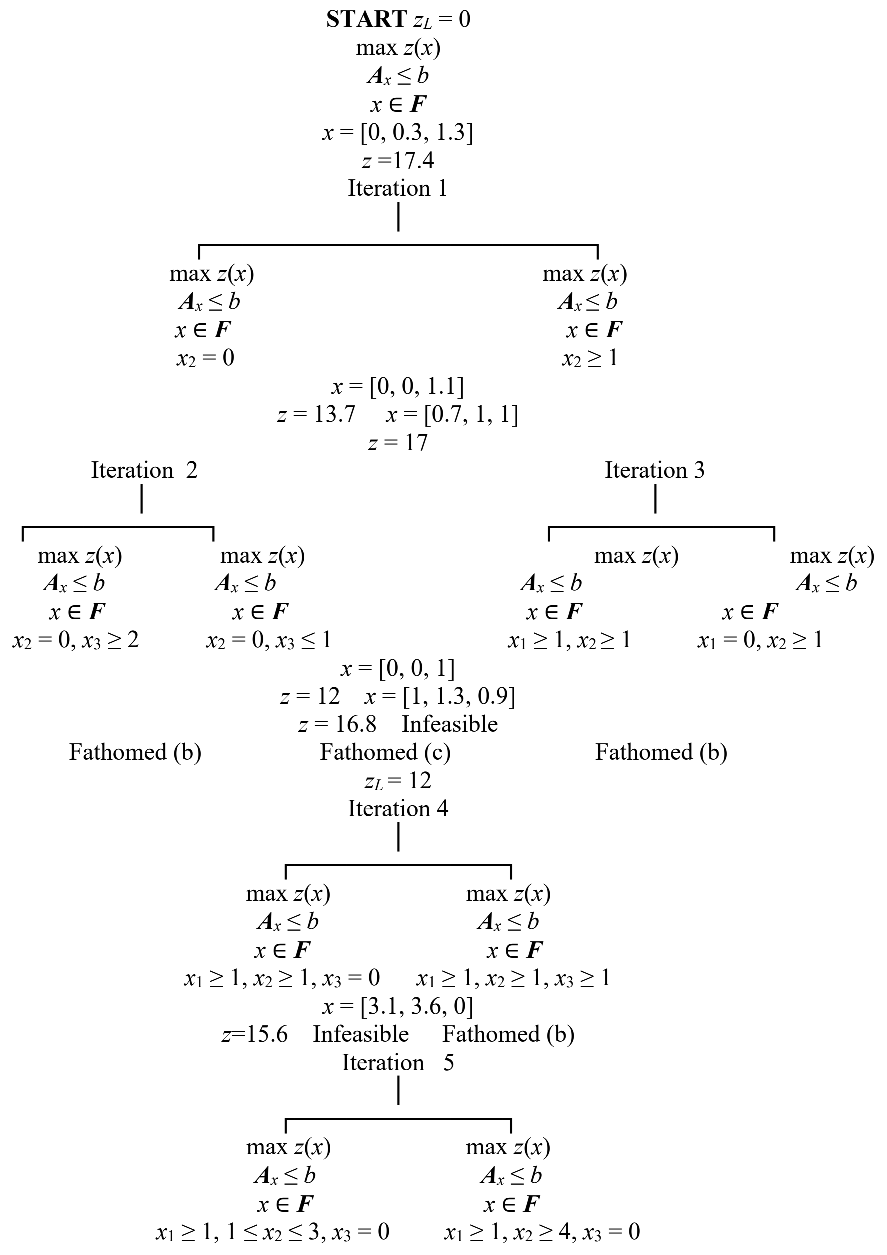


Figure 6.5 Branching Diagram through Iteration 5

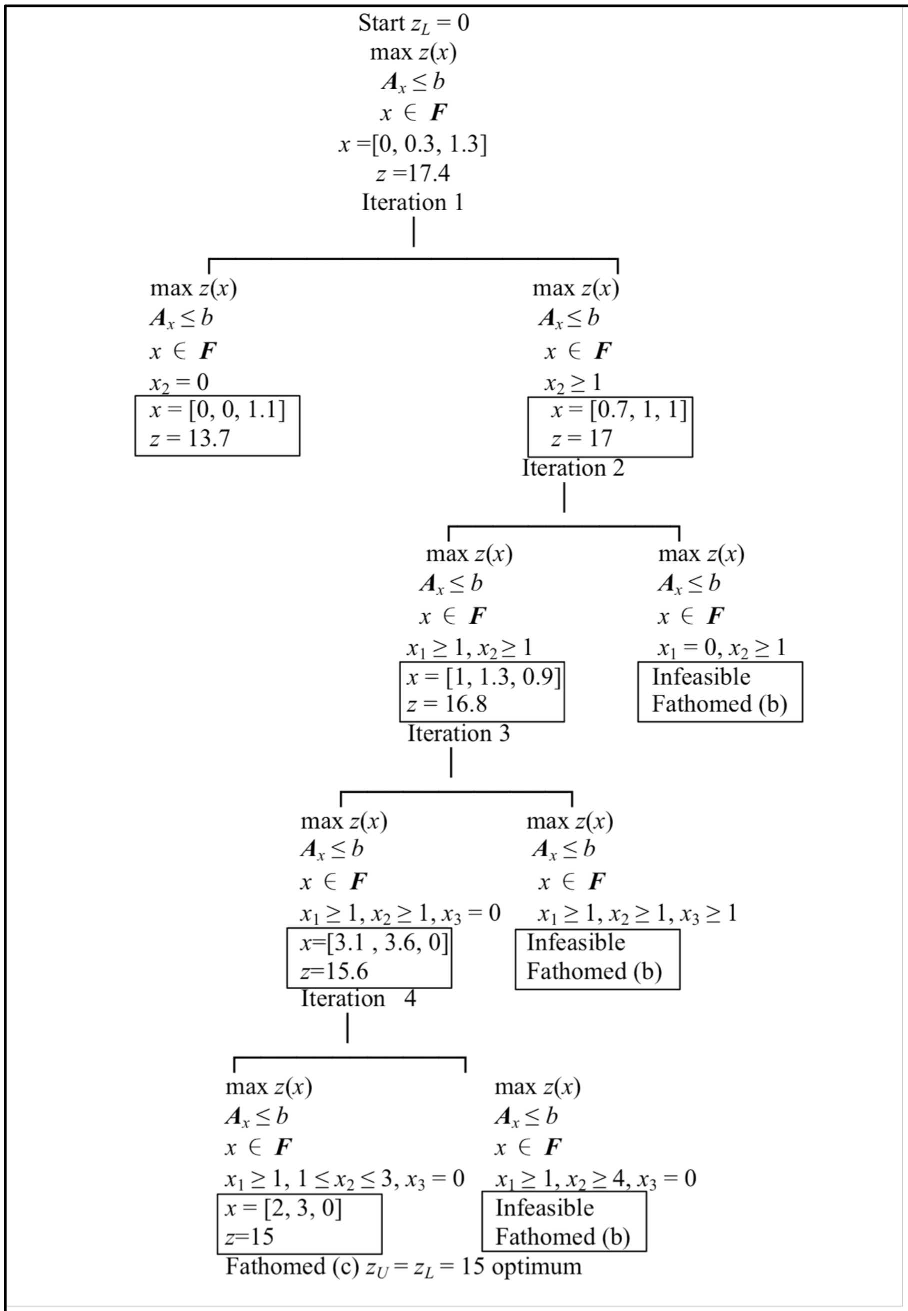


Figure 6-6-Branching Diagram Depth First Subset Selection Strategy

Guidelines and Practical Considerations

The time required to solve a particular problem depends on the way it is formulated. The solution time can be reduced considerably by selecting the variables on which to branch as well as selecting the nodes on which the next branching is to be done.

The choice of branching variables for improved performance are based on factors such as selecting a variable that has the highest fractional value, or a variable that has the greatest importance (which represents an important decision) in the model or the one with the lowest index value [3]. Similarly, the selection of nodes for further branching is based on selecting a node whose LP optimal value is the largest (for maximization problems). In some problems, it might be satisfactory to stop the branch-and-bound algorithm when a solution is within say 3% of the linear programming relaxation of the problem. Also, a tight lower bound on the integer variables helps in reducing the computation time. In addition, the number of integer variables should be as small as possible. This can be done by approximating integer variables that are expected to have large values as continuous variables.

A Branch and Bound Algorithm for Binary Integer Programs

Binary Integer problems can be solved by using the same algorithm described in the previous section. The first step of the algorithm was to solve the linear programming relaxation of the original problem. The resulting solution satisfied the linear inequalities but not the integer restrictions of the original problem. One of the resulting non-integer variables was selected for further branching at the beginning of iteration 1.

However, there exists a different relaxation of the original problem, whose solution yields faster results. This is because, if the integer-programming problem has only 0-1 variables, the bounding step in the branch-and-bound algorithm can be simplified considerably. Because most of the work of the branch-and-bound algorithm is in the bounding step, this simplification can make the algorithm vary much faster. Unlike the one used for the previous example, this relaxation ignores the inequality constraints and requires the variables to be integers. This is in contrast to the previous algorithm that ignored the integer restrictions. This results in a solution set that satisfies the binary (integer) requirements but may not satisfy the inequalities.

The algorithm for this relaxation is obtained by slightly modifying the previous algorithm and is repeated here for maximizing the objective function.

Step 0. *Initialize*

Find an upper bound (for maximization problems) z_U on the objective function. This is done by setting the variables in the objective function with negative coefficients to zero and positive coefficients to one. Check if this solution set satisfies the constraints. If not, then further branching is required. Find a lower bound z_L on the objective function by setting all the variables with positive coefficients to zero and the rest to one. This is the minimum value that the objective function can have, and the optimal solution cannot be lower than this value. This forms the initial lower bound for the problem. Later, as the solution proceeds, the lower bound will be updated if any solution set is found that satisfies all the constraints and has a higher objective function value.

Step 1. *Branch*

Select any (remaining) binary variable to branch on and form two new subsets by setting this binary variable to one and zero respectively.

Step 2. *Bound*

With this variable fixed, find an upper bound on the objective function by setting the remaining variables to 1's and 0's (depending on whether they have positive or negative coefficients). Also find a lower bound on the objective function.

Step 3. *Fathom*

Examine the nodes and fathom a node:

(a) If $z_U \leq z_L$ i.e., The upper bound of the subset is lower than the current lower bound of the problem.

(b) If any of the constraint becomes infeasible as a result of fixing the branching variable to 1 or 0. The way to do it is to find the maximum or minimum value that each constraint can assume (after fixing the branching variables to particular values of 1's or 0's) and to check if it still lies within the limits of the inequalities. If it does not, then it means that there does not exist any combination of 1's and 0's for the remaining variables that can satisfy the constraint. The subproblem is considered to be infeasible if any of the constraint cannot be satisfied, and hence such nodes are fathomed.

(c) If z_U is a feasible solution. If so, then this set is the new incumbent solution since it is the best integer solution obtained so far.

Step 4. *Test*

Return to step 1 if there are any unfathomed nodes. Else, the current lower bound (incumbent solution) is the optimal value for the 0-1 problems.

A major part of this algorithm is the same as given in the previous section. The difference lies in the way the algorithm is implemented i.e., the way in which the subset is determined to be infeasible or not. Also, the upper and lower bounds are obtained by direct substitution. Hence, LP relaxation of the subproblems need not be solved at each step. This results in a faster and easier way to solve binary problems.

A Branch-And-Bound Example for Binary (0-1) Integer Problem

Consider the following example, after Ecker and Kupferschmid [2]. It demonstrates how the speed of branch-and-bound algorithm can be increased when an integer program contains only 0-1 variables by using the algorithm given above.

$$\begin{aligned} \text{max:} \quad & z(x) = 3x_1 + 2x_2 + 5x_3 + 7x_4 && \text{(6-8a)} \\ \text{subject to:} \quad & 3x_1 - 2x_2 + 2x_3 + 5x_4 \geq 6 && \\ & 5x_1 + 3x_2 - x_3 + 4x_4 \leq 3 && \text{(6-8b)} \\ & x_j = 0 \text{ or } 1, j = 1,.....4 && \end{aligned}$$

Relaxation of the above problem after ignoring the inequalities is given below.

$$\text{max:} \quad z(x) = 3x_1 + 2x_2 + 5x_3 + 7x_4 \qquad \text{(6-9)}$$

$$x_j = 0 \text{ or } 1, j = 1...., 4$$

All the objective function coefficients happen to be positive, so the largest possible value of $z(x)$ (upper bound) would be $z(x) = 17$ when $x = [1, 1, 1, 1]$, $z_U = 17$. No further computations would have been needed if this point were to be feasible. It is found by inspection of the original problem that $x = [1, 1, 1, 1]$ is not feasible because the second constraint is violated. In this way, the initial relaxed problem can be solved by inspection. The constraints are then evaluated to decide if this solution is feasible for the original problem.

Next, a lower bound on the objective function value is found by inspection and for the above problem, the minimum value that $z(x)$ could have is when all the nonnegative cost coefficients in the objective function are 0, which yields $z(x) = 0$ when x = [0, 0, 0, 0]. Thus, the branch-and-bound process starts for the above process with $z_L = 0$. The results of the process are shown at the top of the branching diagram in Figure 6-7. The main idea here too is to systematically eliminate from further consideration, subsets that can be determined not to contain the optimal point for the original problem.

The term $x = 0$ _ _ _ and $x = 1$ _ _ _ implies that the value of x_1 is fixed whereas x_2, x_3 and x_4 are free to assume values of either 0 or 1. Hence, this term is referred to as the partial solution of x_1. When a particular combination of x_2, x_3 and x_4 fills up the blanks in a partial solution, the resulting vector is called a completion of the partial solution. A completion that has all zeroes after the partial solution is called the zero completion. For example, 1 0 0 0 is called the zero completion of the partial solution 1 _ _ _. Finally, a completion that satisfies all the inequality constraints is called a feasible completion.

Selecting x_1 as the variable on which to branch, we get two possible configurations;
$x = 1$ _ _ _ and $x = 0$ _ _ _, as shown in Figure 6-7. The next step is to find an upper bound on the objective value for each of the subproblems. As mentioned before, the largest value of the objective function can be found by setting all variables with positive coefficients to one and those with negative coefficients to zero. In this case, the upper bound on the objective function for the right subproblem (with $x_1 = 0$) is $z(x) = 14$, at the point $x = [0,1,1,1]$ whereas for the left subproblem (with $x_1 = 1$); $z_U = 17$ at the point $x = [1,1,1,1]$.

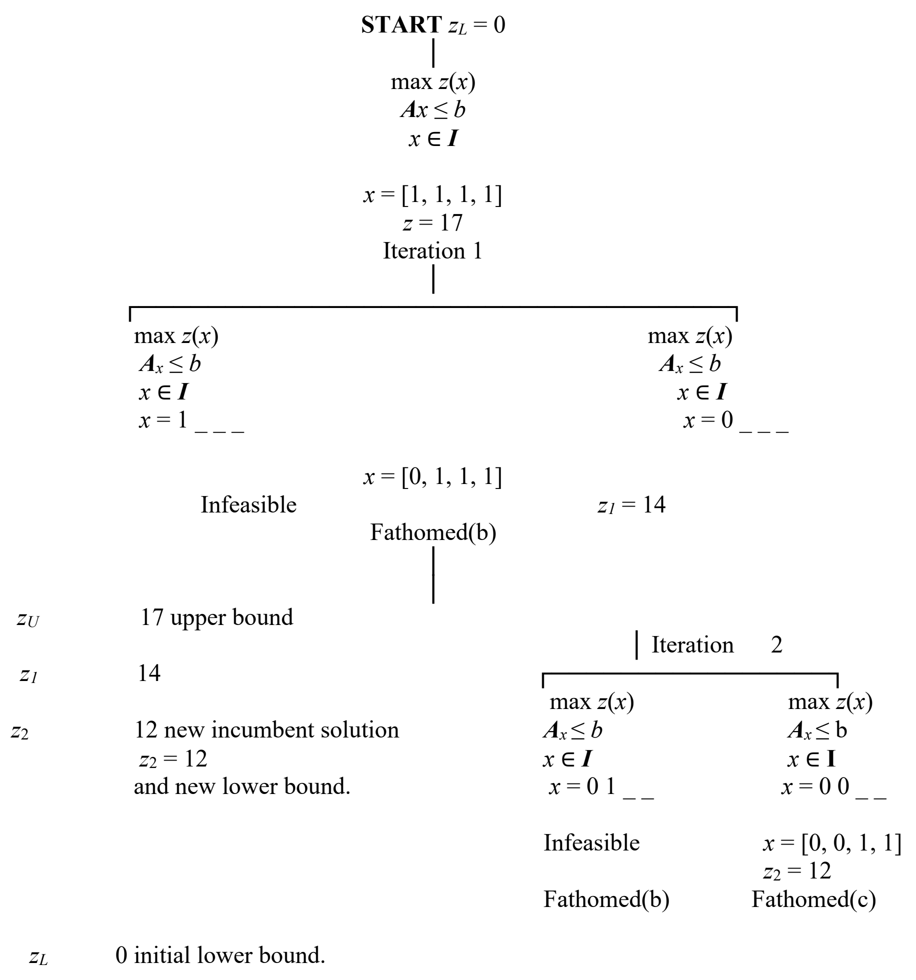


Figure 6-7 Complete Branching Diagram for the Binary Integer Example.

Now, the nodes are subjected to the fathom checks to determine if they could be fathomed. As both subproblems have their upper bound z_U greater than the current lower bound of $z_L = 0$, neither of them can be fathomed on the basis of condition (a). Condition (b) is a check on the feasibility of the subproblems. To determine this, one must find out if there are any feasible completions to the partial solutions of the subproblems. One way of doing this is to exhaustively enumerate all the possibilities for a particular partial completion and check them in the original inequality constraints. In this case, the partial completion $x = 0$ _ _ _ has 8 combinations of completions such as 0000, 0001, 0010, 0011, ..., 0111. This is not practical, especially when there are many variables and constraints.

An alternate and easier method is to simply eliminate subproblems that are discovered to be infeasible. As mentioned before, this is found by checking if the maximum or minimum values of the constraints lie within the limits of their inequalities. The inequality constraints with x_1=1 for the left subproblem are as follows:

$$f_1(x) = 3(1) - 2x_2 + 2x_3 + 5x_4 \geq 6 \qquad (6\text{-}10)$$
$$f_2(x) = 5(1) + 3x_2 - x_3 + 4x_4 \leq 3$$

The largest value that the first constraint can have is obtained by setting all (remaining) variables with positive coefficient to one and the rest to zero. This yields $f_1(x) = 10$ at the point $x = [1,0,1,1]$. Similarly, the minimum value that the second constraint can have is $f_2(x) = 4$ at the point $x = [1,0,1,0]$. Since, even the minimum value of the second constraint cannot satisfy the inequality condition, it implies that there are no completions of the partial solution $x = 1$ _ _ _ that will satisfy the second constraint. This means that x_1 cannot have a value 1 in the final optimal solution. Hence the left node can be fathomed by condition (b). There is no need to check the rest of the constraints if any other constraint is unsatisfied.

Performing the same sort of analysis for the right subproblem with $x_1 = 0$ gives the constraints:

$$f_1(x) = (0) - 2x_2 + 2x_3 + 5x_4 \geq 6 \qquad (6\text{-}11)$$
$$f_2(x) = (0) + 3x_2 - x_3 + 4x_4 \leq 3$$

The first constraint has the maximum value $f_1(x) = 7$ at the point $x = [0,0,1,1]$. Similarly, the second constraint has a minimum value $f_2(x) = -1$ at the point $x = [0,0,1,0]$. Hence, there is at least one completion each for the partial solution $x = 0$ _ _ _ that satisfies the two constraints. Therefore, the right subset cannot be fathomed on the basis that it is infeasible [condition (b)]. This does not necessarily mean that the subset is feasible as there might not be any single completion that satisfies both constraints. Also, no attempt is made to find a single completion that satisfies all constraints.

To finish the fathom check it is necessary to determine whether the point yielding the upper bound on the objective function (in this case, $x = [0,1,1,1]$) is feasible or not. Inspecting the constraints, it is found that the first constraint is not satisfied, and this point is infeasible. Thus, condition (c) fails as well and hence, another branching will be required.

Selecting x_2 as the next variable to branch on, two new subsets are generated with partial solutions 0 1 _ _ and 0 0 _ _. Once again, the first step would be to find an upper bound on the objective value over each of the two new subproblems. For the left subproblem, the upper bound on the objective function is $z(x) = 14$ at the point $x = [0,1,1,1]$ whereas the right subproblem yields an upper bound $z(x) = 12$ obtained at the point $x = [0,0,1,1]$.

Performing the fathom check on the subproblems, we find that the nodes cannot be fathomed by condition (a) because both upper bounds are greater than the current lower bound of $z_L = 0$. Checking for the feasibility of the subproblems, we see that the partial solution 0 1 _ _ of the left subproblem cannot satisfy the first constraint. Hence, the left node is fathomed by condition (b). For the right subproblem, each constraint has feasible completions to the partial solution 0 0 _ _ node and so it cannot be fathomed by condition (b).

The final fathom condition checks the feasibility of the point yielding the upper bound. It turns out that $x = [0,0,1,1]$ satisfies both the constraints and hence, this node is fathomed by condition (c) and the point $x = [0,0,1,1]$ is declared to be the new incumbent solution with $z_L = 12$ as the new lower bound. Finally, as there are no more nodes to be fathomed, $x = [0,0,1,1]$ is declared to be the optimal point with $z = 12$. The final branching diagram is as shown in Figure 6-7.

It should be noted here that when checking for infeasibilities, no attention is paid to the objective function value. Similarly, when an upper bound is being established on the objective function, the constraints are ignored altogether. Moreover, it is never attempted to find the best feasible completion to a subproblem in any single step of the algorithm. This makes each step in the branch-and-bound algorithm easy enough to be performed by inspection for problems that could be worked out by hand.

Mixed Integer Linear Programming

Problems in which only some of the variables assume integer values and the rest are continuous are called as mixed integer programming problems. The integer variables can be either pure integer or binary integer or both. Suppose there are n variables out of which h are integer variables; the mathematical model in the minimization form can be expressed as:

$$\text{Minimize: } z = \sum_{j=1}^{n} c_j x_j, \tag{6-12}$$

$$\text{Subject to: } \sum_{j=1}^{n} a_{ij} x_j < b_i, \quad \text{for } i = 1,2, ..., m,$$

$$\begin{aligned} &x_j \text{ integer} && \text{for } j = 1,2,....,h \ (h \leq n) \\ &x_j \geq 0 && \text{for } j = h + 1, ,...., n \end{aligned}$$

This model becomes a pure integer-programming problem when h is equal to n.

A Branch-And-Bound Algorithm for Mixed Integer Linear Programs

The simplest way of solving mixed integer problems is to use the branch and bound algorithm for general integer programs with the only difference being in the branching step. Though all variables are included in the LP subproblems, branching is done only on integer variables. This ensures that the solution found by the algorithm is optimal for the mixed integer problem. The steps are described as follows for maximizing the objective function.

Step 0. *Initialize.*

Solve the linear programming relaxation of the original problem. If the resulting solution has integer values for all integer variables then the optimal solution for the integer program has been found.

The linear programming relaxation solution provides an upper bound z_U to the problem because the optimal integer solution cannot have an objective function value larger than the linear programming relaxation solution. The imposition of integer restrictions on the integer variables can only make the solution worse.

If the solution does not have integer values for all integer variables, then a lower bound z_L for the optimal objective function value is found that is equal to the objective value at some point that is feasible for the integer program. This could be where all of the variables are zero or some comparable solution that satisfies all the constraints and which will surely be smaller than the final optimal value. If no such feasible point is readily known, set $z_L = -\infty$.

This lower bound solution is also designated as the incumbent solution. This means that it is the best integer solution obtained so far. When a better integer feasible point is obtained as the solution proceeds, then that would be the new incumbent solution.

Step 1. *Branch*

Select an integer variable that currently has a non-integer value from Step 0 and partition the set into two smaller subsets. A subset is obtained from a set by introducing an additional constraint to the set. The additional constraint depends on the integer variable that is selected for branching and also on the (non integer) value of the integer variable when it is selected for further branching.

For example, if the integer variable x_j has the value $k < x_j < k+1$ where k is an integer, then the partitioning is done by adding the constraint $x_j \leq k$ and $x_j \geq k+1$ to the two subsets respectively.

Step 2. *Bound.*

Solve the linear programs that are obtained by appending the extra constraint as a result of Step 1, to the original programming relaxation. These are designated as subsets, and their resulting optimal values (if they are not infeasible) would be the upper bound z_U for that branch when the subset is developed because additional integer constraints are added in expanding branches.

Step 3. *Fathom.*

Examine the subsets that contain the optimal points, and fathom a subset if:

(a) $z_U \leq z_L$, i.e. subset objective function value is less than the lower bound, then no further evaluations are needed.

(b) the subset has no feasible points, then no further evaluations are needed.

(c) If the optimal solution obtained has integer values for all x_j's for $j = 1, 2, \ldots h$ and $z_U > z_L$, then this solution is called the integer-feasible point. It is designated the new incumbent solution, and let $z_L = z_U$.

Step 4. *Test.*

Select a subset among those from Step 1 that have non-integer values for branching. If all subsets have been fathomed, the incumbent solution is optimal for MILP. Otherwise, return to Step 1.

The procedure would remain unchanged even if the objective was to minimize rather than maximize the objective function except that the roles of the upper and lower bounds are reversed. Thus, z_L would be replaced by z_U and vice versa, ∞ becomes $-\infty$, and the directions of the inequalities would be reversed.

Mixed Integer Linear Programming Problem

Consider the following MILP problem, after Murty (7):

$$\max \quad z(x, y) = -3x_2 - 4x_3 - 5x_4 - 20 \tag{6-13a}$$

$$\text{subject to} \quad \begin{aligned} & x_1 - x_2 + x_3 + x_4 = 4 \\ & y_1 + x_2 - 2x_3 + x_4 = 3/2 \\ & y_2 + 2x_2 + x_3 - x_4 = 5/2 \\ & y_1, y_2 \geq 0 \text{ and integer; } x_1 \text{ to } x_4 \geq 0 \end{aligned} \tag{6-13b}$$

The first step is to solve the LP relaxation of the original problem. As shown in the branching diagram of Figure 6-8, the optimal solution does not satisfy the integer requirements of y_1 and y_2. Hence further branching is required on either y_1 or y_2. Selecting y_2, the additional constraints over the two new subsets would be $y_2 \leq 2$ and $y_2 \geq 3$ respectively. The upper bound on the objective value is -20. The right subproblem is solved with this additional constraint and the output is an integer solution with $z_U = -90/4$. Hence this subset is fathomed by condition(c) and the solution declared to be the new incumbent solution with $z_L = z_U$.

Solving the LP relaxation of the left subproblem yields an upper bound of -83/4 which is larger than the current lower bound of $z_L = -90/4$. Hence this subset cannot be fathomed, and further branching is required. Branching on y_1 and solving the right-hand subset, we get an integer solution with $z_L \geq z_U$. This subset is therefore fathomed by condition(c) and the solution is the new incumbent solution. The lower bound is reset to $z_L = -86/4$. Solving the left subproblem and checking the fathom conditions, we find that it can be fathomed by condition(a). Since there are no more subproblems left, the current incumbent solution is the optimal solution for the MIP problem with an objective value $z = -86/4$ and $(y_1, y_2, x_1, x_2, x_3, x_4) = (2, 2, 19/5, 1/10, 3/10, 0)$.

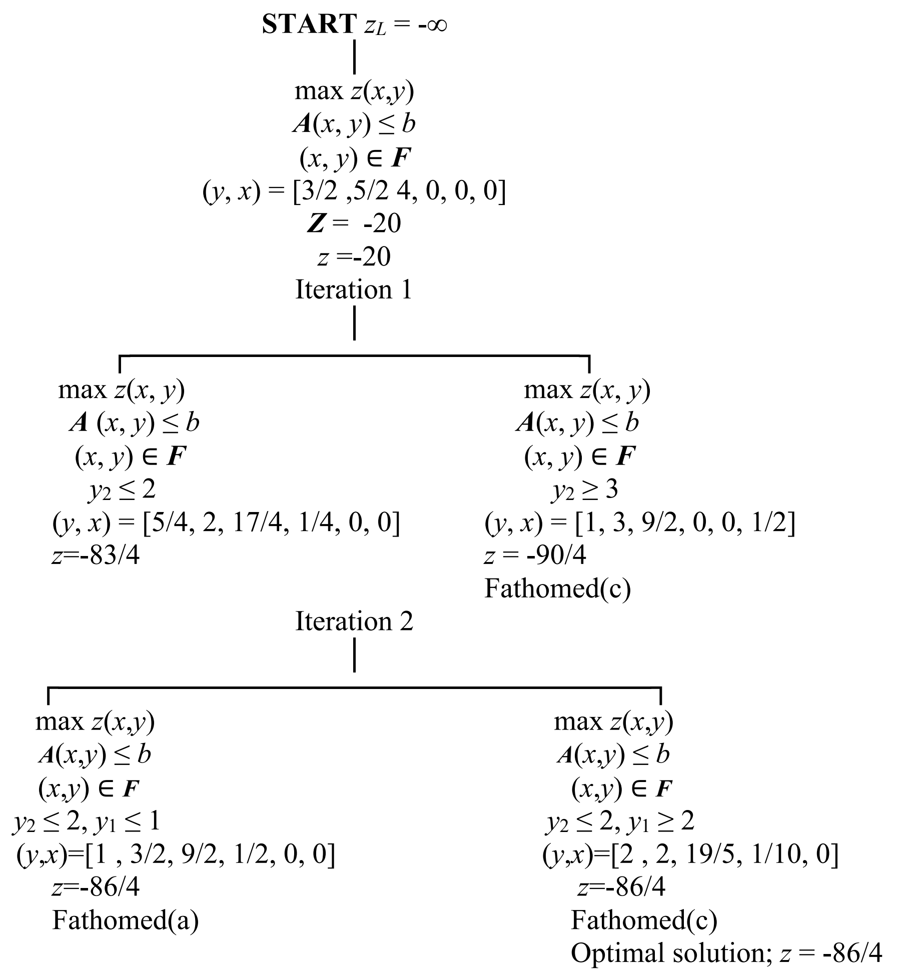


Figure 6-8 Branching Diagram for the MILP Problem

Optimal Process Synthesis and Design

Determining the optimal configuration when designing processes and plants is one of the more important applications of mixed integer programming. This consists of selecting the best configuration of reaction and separations units and the best operating conditions to convert raw materials into products. A superstructure of possible reactors, separators and related units are synthesized, typically using a flowsheeting program. The continuous variables represent the continuous variables such as flow rates, temperature, pressures, and binary variables represent the configuration of process units. The optimal structure and operating conditions are determined by solving a mixed integer linear programming problem (MILP) or a mixed integer nonlinear programming problem (MINLP), depending on the complexity of the process model.

Other industrial applications include heat exchanger synthesis where the optimum heat exchanger network is determined to minimize annual cost and to satisfy the utilities requirements (steam and cooling water) in a plant design or in an existing plant retrofit. Similar results are obtained for mass exchanger networks, chemical reactor networks, distillation column networks and the optimum location for the feed tray in a distillation column to meet product specifications and maximize profit. In batch process scheduling, the optimum sequence for the use of equipment to produce multiple products is determined. In reaction path synthesis, the optimal path is determined to go from raw materials to products, e.g., the manufacture of acetone from ethanol and methane.

To predict the chemical and phase equilibrium for a set of gas, liquid and solid reactants by free energy minimization, the optimization of a mixed integer programing problem is required since the gas, liquid and solid phases may not have all of the components in all of the phases. There are a number of special programs to perform this evaluation that contain extensive thermodynamic properties in polynomial form.

Following the description of methods to formulate mixed integer problems, an example is given for a process superstructure where the optimum structure is obtained by solving a MILP using the optimization program GAMS. The important feature of the GAMS is that this optimization programming language uses the same structure and format that is used to express the optimization problem mathematically, and there are a number of solvers that can be called to preform the optimization, depending on the type of problem.

Summary of MIP Problem Formulations

In formulating the optimization problem, a convention is used. In selecting among process units, the following equations are used with integer variable y_i where y_i is 1 if process i is selected and 0 if not.

$\sum y_i = 1$ select only one unit

$\sum y_i \leq 1$ select at most one unit

$\sum y_i \geq 1$ select at least one unit

$y_j - y_i \leq 0$ select unit i only if unit j is selected

The last condition is used when there are several sequences of process units from which one sequence is to be selected.

For activation or deactivation of continuous variables, the bounds on capacities on a process unit can be used. If a process unit does not exist, then the inlet flow rate should be zero; and if it exists, the flow rate should be within the bounds of the upper and lower limits, F_{iL} and F_{iU} . This can be expressed as:

$F_{iL}\, y_i \leq F_i \leq F_{iU}\, y_i$

for $y_i = 1$ then $F_{iL} \leq F_i \leq F_{iU}$

$y_i = 0$ then $0 \leq F_i \leq 0$ or $F_i = 0$

For activation and relaxation of constraints, consider the constraints $f_1(\mathbf{x}) = 0$ and $f_2(\mathbf{x}) \leq 0$ that describe a process unit. If the process unit exist, then $y_i = 1$, and the constraints should be active. If the process unit does not exist, then $y_i = 0$, and constraints should be do not exist (are inactive). This case can be formulated using slack variables s_1, s_2 and s_3 and upper bounds U_1 and U_2. Slack variables are variables that are added to inequality constraint equations to convert them to equality constraints.

$$
\begin{aligned}
&f_1(\mathbf{x}) + s_1 - s_2 = 0 \\
&f_2(\mathbf{x}) \leq s_3 \\
&s_1 + s_2 \leq U_1 (1 - y_i) \\
&s_3 \leq U_2 (1 - y_i)
\end{aligned}
$$

and s_3 can be eliminated to give:

$$f_2(\mathbf{x}) - U_2 (1 - y_i) \leq 0$$

For example, if $y_i = 1$, then $s_1 - s_2 \leq 0$ or $s_1 - s_2 = 0$, and $s_1 = s_2 = 0$ since both are positive. This gives the result that:

$$
\begin{aligned}
&f_1(\mathbf{x}) = 0 \\
&f_2(\mathbf{x}) \leq 0
\end{aligned}
$$

If $y_i = 0$, then $s_1 + s_2 \leq U_1$ and $s_3 \leq U_2$. The constraints are inactive.

$$
\begin{aligned}
&f_1(\mathbf{x}) + s_1 - s_2 = 0 \\
&f_2(\mathbf{x}) \leq s_3 \leq U_2
\end{aligned}
$$

This and additional information are given by Floudas, (19) for nodes with several inputs, logical constraints and bilinear products.

Example for Optimal Design of a Chemical Complex

In this example, modified from Karimi (6), a mixed integer-programming problem is solved to demonstrate the selection of the optimal process design from options to make or purchase raw materials for the plant. The diagram in Figure 6-9 shows a superstructure of several options to produce the product from the raw materials.

As shown in Figure 6-9, a company is evaluating producing chemical C (propylene oxide) from B (propylene) in either Process 2 (chlorohydrin process) or Process 3 (peroxide process). Also, B (propylene) can be made in Process 1 (steam cracking of propane to propylene) using A (propane) as a raw material, or B (propylene) can be purchased from another company.

This evaluation requires solving a mixed integer linear programming problem. The economic model includes fixed and operating costs as given in the table below. The constraints are material balances mass yields, demand for product and availability of raw materials as shown in the table. Integer variables are used to have C produced from B in either process 2 or process 3 and to have B either produced in process 1 or purchased from another company.

The optimal solution will select either process 2 or 3 to produce C and determine if B is to be purchased or produced in process 1 by maximizing the profit. Also, the optimal amounts of B and C will be determined given the demand for C and the availability of A.

Economic Data

Process	Fixed Cost ($/hr)	Operating Cost ($/ton of feed)	Feed Cost ($/ton)		Sales Product Price ($/ton)	
1	1,000	250	A	500	C	1,800
2	1,500	400	B	950		
3	2,000	550				

Process Data

Process	Mass Yields	Demand for Product	Availability of Raw Materials
1 (A to B)	0.90	$C \leq 10$ tons/hr.	$A \leq 16$ tons/hr
2 (B to C)	0.82		
3 (B to C)	0.95		

The process variables are defined as follows where F designates the mass flow rate in tons per hour. The first subscript specifies the stream number and the second subscript gives the component (chemical species) in the stream.

F_{1A}	flow rate of A to Process 1
F_{2B}	flow rate of B to either Process 2 or 3 if Process 1 is selected
F_{3A}	flow rate of unreacted A from Process 1
F_{4B}	flow rate of B purchased from a supplier if a supplier is selected
F_{5B}	flow rate of B to either Process 2 or 3
F_{6B}	flow rate of B to Process 2 if Process 2 is selected
F_{7B}	flow rate of B to Process 3 if Process 3 is selected
F_{8C}	flow rate of C if Process 2 is selected
F_{9B}	flow rate of unreacted B if Process 2 is selected
F_{10C}	flow rate of C if Process 3 is selected
F_{11B}	flow rate of unreacted B if Process 3 is selected
F_{12C}	flow rate of C to sales

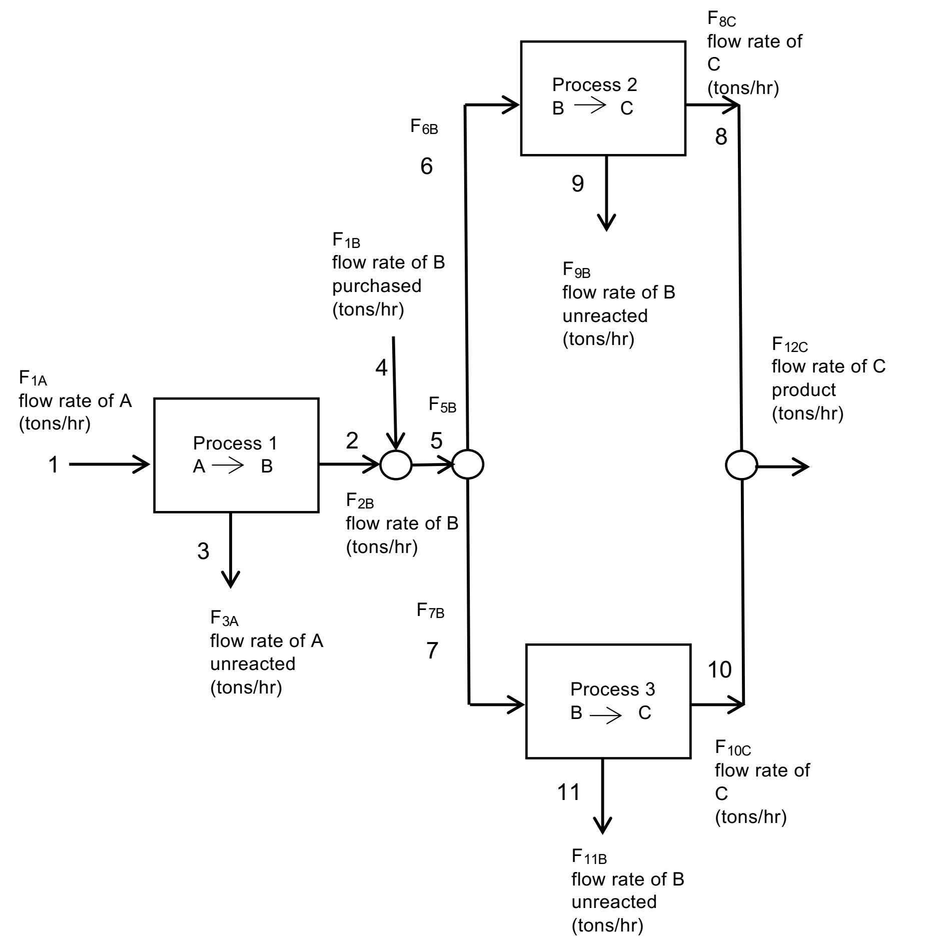


Figure 6-9 Process Superstructure Diagram for Optimal Design Example

Integer variables are used to ensure either process 1 is used for making B from A or B is purchased. Also, they are used to ensure that either Process 2 or 3 is selected. They are defined as follows:

y_1 = 1 if Process 1 is selected and 0 if not
y_2 = 1 if Process 2 is selected and 0 if not
y_3 = 1 if Process 3 is selected and 0 if not
y_4 =1 if B is purchased and 0 if not

The material balances associated with the processes and the nodes in the diagram are as follows.

Conversion of A to B in Process 1:

$F_{2B} = 0.90\ F_{1A}$
$F_{3A} = 0.10\ F_{1A}$

Conversion of B to C in Process 2:

$F_{8C} = 0.82\ F_{6B}$
$F_{9B} = 0.18\ F_{6B}$

Conversion of B to C in Process 3:

$F_{10C} = 0.95\ F_{7B}$
$F_{11B} = 0.05\ F_{7B}$

Material balance on B at node between processes:

$F_{2B} + F_{4B} = F_{5B}$
$F_{5B} = F_{6B} + F_{7B}$

Material balance on C at the node from Processes 2 and 3:

$F_{8C} + F_{10C} = F_{12C}$

Availability of raw material A:

$F_{1A} \leq 16$ must be modified to include the possibility of not having Process 1
$F_{1A} \leq 16\ y_1$ operating by incorporating binary integer variable y_1

Availability of raw material B:

$F_{4B} \leq 20$ must be modified to include the possibility of only purchasing B
$F_{4B} \leq 20\ y_4$ by incorporating binary integer variable y_4

Demand for product C:

$F_{12C} \leq 10$ must be modified to include the possibility of only having Process 2 or 3
$F_{8C} \leq 10\ y_2$ operating by incorporating binary integer variables y_2 and y_3
$F_{10C} \leq 10\ y_3$

Integer equations

Integer equation forcing the selection of Process 1 or purchase of B
$y_1 + y_4 = 1$

Integer equation forcing the selection of either Process 2 or 3
$y_2 + y_3 = 1$

Combining the constraint equations with the economic model in the MILP format gives:

$$\underbrace{-250F_{1A} - 400F_{6B} - 550\,F_{7B}}_{\text{operating cost}} \underbrace{- 1{,}000y_1 - 1{,}500y_2 - 2{,}000y_3}_{\text{fixed cost}} \underbrace{-500\,F_{1A} - 950\,F_{4B}}_{\text{feed cost}} \underbrace{+ 1{,}800\,F_{12C}}_{\text{sales}}$$

max: $-250F_{1A} - 400F_{6B} - 550\,F_{7B} - 1{,}000y_1 - 1{,}500y_2 - 2{,}000y_3 -500\,F_{1A} - 950\,F_{4B} + 1{,}800\,F_{12C}$

subject to: mass yields

$$\begin{aligned} -0.90\,F_{1A} + F_{2B} &= 0 \\ -0.10\,F_{1A} + F_{3A} &= 0 \\ -0.82\,F_{6B} + F_{8C} &= 0 \\ -0.18\,F_{6B} + F_{9B} &= 0 \\ -0.95\,F_{7B} + F_{10C} &= 0 \\ -0.05\,F_{7B} + F_{11B} &= 0 \end{aligned}$$

node MB

$$\begin{aligned} F_{2B} + F_{4B} - F_{5B} &= 0 \\ F_{5B} = F_{6B} - F_{7B} &= 0 \\ F_{8C} + F_{10C} - F_{12C} &= 0 \end{aligned}$$

availability of A $\quad F_{1A} \leq 16\,y_1$

availability of B $\quad F_{4B} \leq 20\,y_4$

demand for C $\quad F_{8C} \leq 10\,y_2$

$\quad F_{10C} \leq 10\,y_3$

integer constraints

$$\begin{aligned} y_2 + y_3 &= 1 \\ y_1 + y_4 &= 1 \end{aligned}$$

The optimal structure for the example was obtained using the GAMS program in Figure 6-10. The start of the results is an echo print of the program as shown in Figure 6-10 that includes defining binary and positive variables and the equations. This is followed by the equations for the process model and the objective function. Statement on line 67 has the program use all of the equations and on line 69 to maximize the PROFIT using the solver MIP. Then the results give a status of the solution including: 1 normal completion, 1 optimum found and the value of the objective function at the optimum. This is followed by values for the lower level, upper and marginal values of the constraint equations. The marginal values are the values of the Lagrange multipliers, the level values are for the inequality constraints, and "." is used to indicate a zero value. See the sensitivity analysis discussion in the linear programming chapter. This is followed by values for the lower level, upper and marginal values of the variables. The variables in the optimum basis will leave the basis if the upper and lower limits are exceeded, as discussed in the section on sensitivity analysis in the linear programming chapter.

Figure 6-10 GAMS Program and Results for the Optimal Design Example

GAMS Program

```
Design of a Chemical Complex
*filename:  PROCESS.gms
option optcr=0, limrow=0, limcol=0;

BINARY VARIABLES
     Y1 denotes selection of process 1 when equal to one
     Y2 denotes selection of process 2 when equal to one
     Y3 denotes selection of process3I when equal to one
     Y4 denotes selection of purchased B when equal to one;

POSITIVE VARIABLES
     F1A    Flow rate of A to Process 1 (All flow rates in tons per hour)
     F2B    Flow rate of B to either Process 2 or 3 if Process 1 selected
     F3A    Flow rate of unreacted A from process 1
     F4B    Flow rate of B purchased from a supplier if supplier selected
     F5B    Flow rate of B to either Process 2 or 3
     F6B    Flow rate of B to Process 2 if Process 2 selected
     F7B    Flow rate of B to Process 3 if process 3 selected
     F8C    Flow rate of C if Process 2 selected
     F9B    Flow rate of unreacted B if Process 2 selected
     F10C   Flow rate of C if Process 3 selected
     F11B   Flow rate of unreacted B if Process 3 is selected
     F12C   Flow rate of C to sales          ;

     VARIABLE PROFIT objective function       ;

     EQUATIONS
     E1     conversion of B to C in Process 1
     E2     unreacted A from mass balance on Process 1
     E3     conversion of C in Process 2
     E4     unreacted B from mass balance on Process 2
     E5     conversion of B to C in Process 3
     E6     unreacted B from mass balance on Process 3;
     E7     material balance on node from Processes 1 and purchased B
     E8     material balance on node to Processes 2 and 3
     E9     material balance on node from processes 2 and 3 to sales
     E10    availability of raw material
     E11    demand for product if from Process 2
     E12    demand for product if from Process 3
     E13    integer constraint to select either Process 1 or purchase B
     E14    integer constraint to select either process 2 or 3
     OBJ    objective function definition;
```

```
44
45       E1 ..    -0.90*F1A  + F2B  =E= 0       ;
46       E2 ..    -0.10*F1A  + F3A  =E= 0       ;
47       E3 ..    -0.82*F6B  + F8C  =E= 0       ;
48       E4 ..    -0.18*F6B  + F9B  =E= 0       ;
49       E5 ..    -0.95*F7B  + F10C =E= 0       ;
50       E6 ..    -0.05*F7B  + F11B =E= 0       ;
51       E7 ..    F2B +  F4B -   F5B     =E= 0  ;
52       E8 ..    F5B -  F6B -   F7B     =E= 0  ;
53       E9 ..    F8C +  F10C-  F12C     =E=0   ;
54       E10..    F1A -  16*Y1           =L= 0  ;
55       E11..    F8C -  10*Y2           =L= 0  ;
56       E12..    F10C - 10*Y3           =L= 0  ;
57       E13..    Y1+    Y4              =L=1   ;
58       E14..    Y2+    Y3              =L=1   ;
59
60  * constraint for the maximum demand of product C
61  * is declared as an upper bound here
62                F12C.UP                       = 10    ;
63
64       OBJ .. PROFIT =E= -250*F1A - 400*F6B - 550*F7B - 1000*y1 -1500*y2
65                - 2000*y3 -500*F1A  - 950*F4B +1800*F12C                    ;
66
67       MODEL PROCESS /ALL/                                                  ;
68
69       SOLVE PROCESS USING MIP MAXIMIZING PROFIT                            ;
```

Printout of Results from the GAMS Program

```
COMPILATION TIME     =       0.000 SECONDS    0.7 Mb      WIN-18-097
Design of a Chemical Complex
Model Statistics          SOLVE PROCESS USING MIP FROM LINE 69

MODEL STATISTICS

BLOCKS OF EQUATIONS            15       SINGLE EQUATIONS       15
BLOCKS OF VARIABLES            17       SINGLE VARIABLES       17
NON ZERO ELEMENTS              40       DISCRETE VARIABLES      4

GENERATION TIME          =       0.000 SECONDS       1.4 Mb           WIN-18-097
EXECUTION TIME           =       0.000 SECONDS       1.4 Mb           WIN-18-097

                          SOLVE SUMMARY
         MODEL   PROCESS                          OBJECTIVE  PROFIT
         TYPE    MIP                              DIRECTION  MAXIMIZE
         SOLVER  OSL                              FROM LINE  69
```

```
**** SOLVER STATUS            1 NORMAL COMPLETION
**** MODEL STATUS             1 OPTIMAL
**** OBJECTIVE VALUE          459.3496

 RESOURCE USAGE, LIMIT        0.160     1000.000
ITERATION COUNT, LIMIT        5         10000

OSL Version 1 Jul 4, 1999 WIN.OS.18.1 055.035.036.WAT OSL Version 1
Work space allocated          --    0.18 Mb

                         LOWER     LEVEL    UPPER    MARGINAL
---- EQU E1              .             .    .           833.333
---- EQU E2              .             .    .           EPS
---- EQU E3              .             .    .           1504.065
---- EQU E4              .             .    .           EPS
---- EQU E5              .             .    .           1456.140
---- EQU E6              .             .    .           EPS
---- EQU E7              .             .    .           -833.333
---- EQU E8              .             .    .           -833.333
---- EQU E9              .             .    .           -1656.140
---- EQU E10             -INF      -2.450   .           .
---- EQU E11             -INF          .    .           152.075
---- EQU E12             -INF          .    .           200.000
---- EQU E13              -INF     1.000    1.000       .
---- EQU E14             -INF      1.000    1.000       .
---- EQU OBJ             .             .    .           1.000
```

E1 conversion of B to C in Process 1
E2 unreacted A from mass balance on Process 1
E3 conversion of C in Process 2
E4 unreacted B from mass balance on Process 2
E5 conversion of B to C in Process 3
E6 unreacted B from mass balance on Process 3;
E7 material balance on node from Processes 1 and purchased B
E8 material balance on node to Processes 2 and 3
E9 material balance on node from processes 2 and 3 to sales
E10 availability of raw material
E11 demand for product if from Process 2
E12 demand for product if from Process 3
E13 integer constraint to select either Process 1 or purchase B
E14 integer constraint to select either process 2 or 3
OBJ objective function definition;

```
                        LOWER       LEVEL       UPPER       MARGINAL
---- VAR Y1             .           1.000       1.000       -1000.000
---- VAR Y2             .           1.000       1.000       20.753
---- VAR Y3             .           .           1.000       EPS
---- VAR Y4             .           .           1.000       EPS
---- VAR F1A            .           13.550      +INF        .
---- VAR F2B            .           12.195      +INF        .
---- VAR F3A            .           1.355       +INF        .
---- VAR F4B            .           .           +INF        -116.667
---- VAR F5B            .           12.195      +INF        .
---- VAR F6B            .           12.195      +INF        .
---- VAR F7B            .           .           +INF        .
---- VAR F8C            .           10.000      +INF        .
---- VAR F9B            .           2.195       +INF        .
---- VAR F10C           .           .           +INF        .
---- VAR F11B           .           .           +INF        .
---- VAR F12C           .           10.000      10.000      143.860
---- VAR PROFIT         -INF        459.350     +INF        .
```

Y1	denotes selection of process 1 when equal to one
Y2	denotes selection of process 2 when equal to one
Y3	denotes selection of process 3 when equal to one
Y4	denotes selection of purchased B when equal to one;
F1A	Flow rate of A to Process 1 (All flow rates in tons per hour)
F2B	Flow rate of B to either Process 2 or 3 if Process 1 selected
F3A	Flow rate of unreacted A from process 1
F4B	Flow rate of B purchased from a supplier if supplier selected
F5B	Flow rate of B to either Process 2 or 3
F6B	Flow rate of B to Process 2 if Process 2 selected
F7B	Flow rate of B to Process 3 if process 3 selected
F8C	Flow rate of C if Process 2 selected
F9B	Flow rate of unreacted B if Process 2 selected
F10C	Flow rate of C if Process 3 selected
F11B	Flow rate of unreacted B if Process 3 is selected
F12C	Flow rate of C to sales;
PROFIT	objective function

```
**** REPORT SUMMARY:
                  0       NONOPT
                  0       INFEASIBLE
                  0       UNBOUNDED
EXECUTION TIME     = 0.000 SECONDS        0.7 Mb          WIN-18-097
```

Computer Codes Available for Solving MILP Problems

MILP models can be solved using a variety of computer codes. A few of which are described here. The main frame mixed integer-programming solver that has been available for a number of years is the IBM Mathematical Programming System Extended (MPSX/370) that supports Mixed Integer Programming (MIP). Since this solver is a mainframe utility, it can handle very large problems. The problem data is stored in MPS file format and a separate Control Program is written to solve the problem. The detailed documentation is given in IBM manuals [9].

GAMS (General Algebraic Modeling System) is a program for solving LP, MILP as well as NLP and MINLP. This system was developed at the World Bank to solve very large economic problems and extended by the GAMS Development Corporation in Washington D. C. GAMS is a high-level language that makes concise algebraic statements of models and hence is easier to understand and implement. Detailed documentation of GAMS is given in the GAMS manual [10]. One of the advantages of GAMS is that the computer code uses the same format as the mathematical statement of the optimization problem.

LINDO, LINGO and "What's Best!" by Lindo Systems Inc. solves both LP and MILP problems. The formulation of models is straightforward, and the user has to list all the constraints one by one. This gets tedious if the constraints in the model are expressed in the summation form because, then each constraint will have to be separately written for the above three solvers. Nevertheless, these solvers are very convenient to use for small problems.

MILP Approach in Batch Plant Scheduling

Scheduling of batch plant operation is a very important application of mixed integer linear programming according to Mah [8], and typically production rate of up to 25 million pounds per year are done in batch plants. Products such as pharmaceuticals, fermentation products, paints, plastics and food products are manufactured in batch processes.

A batch plant may be used to produce a single product or multiple products using the same set of equipment. Several products need the same processing steps that pass through the same series of processing units, and these are called multiproduct plants. As batches of different products require different processing times, the total time required to produce a set of batches depends on the sequence in which they are produced. Hence, it is crucial to schedule the batch operations in such a way so as to maximize plant productivity by minimizing the total time required to complete the entire set of operations (called the makespan).

Multiproduct batch plant scheduling problems can be solved using mixed integer linear programming. Problems of this type can be considered to be consisting of two interlinked subproblems. The first one is the determination of the order in which the products are to be produced, and the second subproblem deals with the determination of the start and finish times of each product on all processing units. The final plant schedule corresponding to a sequence is then represented in the form of a Gantt chart [8].

Optimal Multi-Batch Batch Scheduling

As an example, the unit processing times for a 4-product, 3-stage flow shop are shown in Table 6-1. The Gantt chart in Figure 6-9 shows one of the possible schedules for the sequence of jobs. From this chart, we see that in Unit 1, there is a wait (holding) time of 5-unit times for product 3. This is because Unit 2 is not yet ready to accept output from Unit 1. Therefore, Unit 1 has to hold product 3 until Unit 2 is ready to accept it. This holding time is shown by the shaded area in the Gantt Chart. Similarly, product 2 in Unit 2 has to wait for 3-time units before Unit 3 becomes ready. The total time required (makespan) for this particular sequence is 27-time units. This is just one of the possible job sequences, and it may be far from optimal. The formulation of this kind of MILP problem involves representing the batch plant scheduling configurations in terms of mathematical equations that are expressed below (8).

Table 6-1 Unit Processing Times for a Four Product, Three-Stage Flow Shop

UNIT	P1	P2	P3	P4
1	5	3	2	5
2	3	4	3	4
3	7	2	7	3

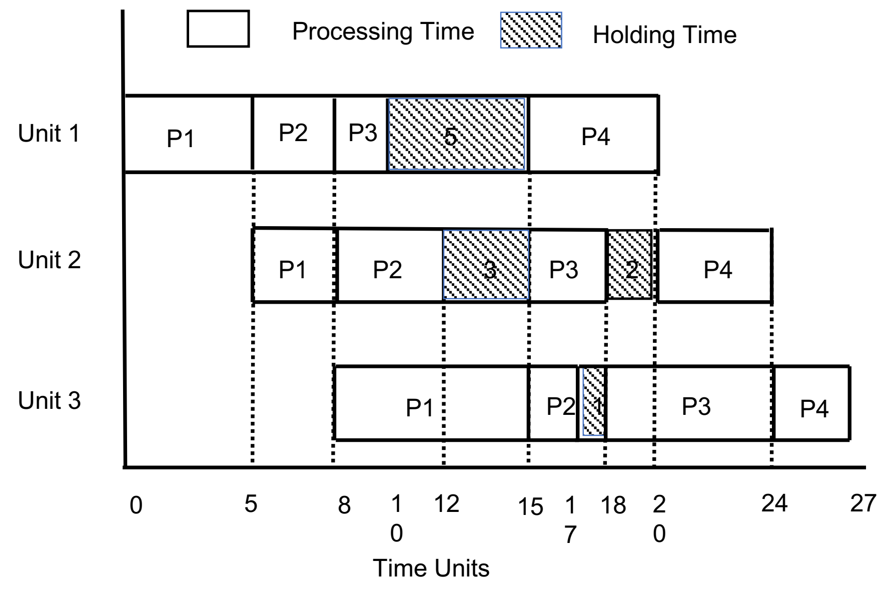


Figure 6-11 Gantt Chart for 4-Product, 3-Stage Flow Shop

A description of this scheduling problem can be formulated as a MILP. Let N be the number of products and M be the number of processing units or stages in a plant. As shown in the Gantt Chart in Figure 6-11, a product i (i = 1..., N) can occupy only one slot j (j = 1..., N) in each unit k (k = 1..., M). This can be expressed mathematically by defining a binary variable y_{ij} such that:

$$y_{ij} = \begin{cases} 1, \text{ if product } i \text{ is placed in slot } j \text{ in the sequence} \\ 0, \text{ otherwise} \end{cases} \qquad (6\text{-}14)$$

The constraint that ensures that each product i is assigned to exactly one position j in the sequence is given by:

$$y_{i1} + y_{i2} + y_{i3} + \ldots + y_{iN} = 1 \quad \text{or} \quad \sum_{j=1}^{N} y_{ij} = 1 \quad \text{for } i = 1, \ldots, N \qquad (6\text{-}15)$$

Similarly, the constraint that ensures that each position in the product sequence is assigned to only one product is given by:

$$y_{1j} + y_{2j} + y_{3j} + \ldots + y_{Nj} = 1 \quad \text{or} \quad \sum_{i=1}^{N} y_{ij} = 1 \quad \text{for } j = 1, \ldots, N \qquad (6\text{-}16)$$

Let C_{ik} be the completion time i.e., the time at which the i^{th} product leaves unit k after completion of its processing. Here, i^{th} product means the product in slot i.

Let the processing time PT_{ik} be the time required to process the i^{th} product in unit k. Now, the i^{th} product cannot leave unit k until it is processed and in order to get processed, it must have left unit (k-1).

Thus, the completion time for product i in unit k i.e., C_{ik} must be at least equal to its completion time of unit (k-1) plus its processing time (PT) in unit k. This can be represented as:

$$C_{ik} \geq C_{i(k-1)} + PT_{ik}, \qquad \text{for } i = 1,\ldots N \text{ and } k = 2 \ldots M \qquad (6\text{-}17a)$$

Equation 9-17a was formulated under the condition that there is at least one unit before unit k and the limits of k are from 2 to M.

Similarly, i^{th} product cannot leave unit k until (i - 1)th product has been processed, and the former has been processed. Therefore,

$$C_{ik} \geq C_{(i-1)k} + PT_{ik}, \qquad \text{for } i = 1,\ldots N \text{ and } k = 1, M \text{ with } C_0k = 0 \qquad (6\text{-}17b)$$

Finally, the i^{th} product can leave unit k only when unit (k +1) is free i.e., when the (i - 1)th product in unit (k+1) has left. This can be represented as:

$$C_{ik} \geq C_{(i-1)(k+1)} \qquad \text{for } i = 1, \ldots N; \quad k = 1,\ldots M \qquad (6\text{-}17c)$$

From Equation 6-17(c), it implies that $C_{i(k-1)} \geq C_{(i-1)k}$ (substituting $k = k - 1$) into Equation 9-17c. Equations 6-17(a) and 6-17(c) imply 6-17(b) for $k = 2, \ldots M$. Now:

$$Cik \geq Ci(k\text{-}1) + PTik \geq C(i\text{-}1)k + PTik.$$

Therefore, Equation 6-17(b) for $k = 2, \ldots M$ are redundant, and Equation 6-17(c) is given for $k = 1$ only.

Let t_{ik} be the processing time for product i. Now, if product i is in slot j, then PT_{jk} must be t_{ik}. Also, for a given unit k, product i can be in only one slot. Thus, y_{ij} can be used to pick the right processing time representing PT_{jk}. This can be represented mathematically as:

$$PT_{ik} = y_{1i}\, t_{1k} + y_{2i}\, t_{2k} + y_{3i}\, t_{3k} + \ldots + y_{Ni}\, t_{Nk}$$

or

$$PT_{ik} = \sum_{j=1}^{N} y_{ji}\, t_{jk} \quad \text{for } i = 1, \ldots N \;\; k = 2, \ldots M \tag{6-18}$$

(9-18)

Substituting Equation 6-18 into Equation 6-17a gives:

$$C_{ik} \geq C_{i(k-1)} + \sum_{j=1}^{N} y_{ji}\, t_{jk} \qquad \text{for } i = 1, \ldots N \;\; k = 2, \ldots M \tag{6-19}$$

The MILP for the batch-scheduling problem is:

Minimize: C_{NM} (6-20a)

Subject to: $C_{ik} \geq C_{i(k-1)} + \sum_{j=1}^{N} y_{ji}\, t_{jk}$ for $i = 1 \ldots N$ $k = 2 \ldots M$ (6-20b)

$C_{i1} \geq C_{(i-1),1} + \sum_{j=1}^{N} y_{ji}\, t_{j1}$ for $k = 2 \ldots M$ (6-20c)

$C_{ik} \geq C_{(i-1)(k+1)}$ for $i = 1, \ldots N$; $k = 1, \ldots M\text{-}1$ (6-20d)

$\sum_{j=1}^{N} y_{ij} = 1$ for $i = 1, \ldots, N$ (6-20e)

N

$$\sum_{i=1} y_{ij} = 1 \quad \text{for } j = 1, \ldots, N \qquad (6\text{-}20f)$$

$C_{ik} \geq 0$, and y_{ij} binary

These equations represent batch plant scheduling problems in which the objective is to find the optimal scheduling sequence for various jobs in order to minimize the makespan, the completion time C_{NM}. Equation 6-15 ensures that each product i is assigned to exactly one position j. Equation 6-16 ensures that each product sequence is assigned to only one product. Equation 6-17a ensures that the completion time for product i in unit k, C_{ik}, is greater than or equal to the completion time of the prior units $C_{i(k-1)}$ and the processing time of product i in unit k, PT_{ik}. Equation 6-17b ensures that the completion time for product i in unit k, C_{ik}, is greater than or equal to the time for the product completion time product of $(i - 1)^{th}$ has and the processing time of product i in unit k, PT_{ik}. These constraints describe the minimum completion times of processing of a particular product i in slot j and unit k.

Additional extensions of the batch processing MINLP by Mah (8) and Ku (14) include limited intermediate storage, no intermediate storage, finite intermediate storage, mixed intermediate storage, and zero weight. A systematic method for batch processing scheduling with limited resources is described by Ku and Karimi (15). A review of continuous-time versus discrete-time approaches for scheduling of chemical processes is given by Floudas and Lin (16)

An example of a no intermediate storage problem given by Karimi in CACHE- Process Design Case Studies, Vol. 6 (6) is described below. A GAMS program for the solution is given in the CD with the Case Studies (6).

Example of a Multiproduct Batch Plant Scheduling Problem

A multiproduct plant wishes to produce four products (P1 - P4) in batches. Each product requires three processing steps that are carried out by three batch units. The processing times of each product for the three units is given in Table 9-2.

Table 9-2. Processing Times (hours) of products, after Karimi [6].

	Products			
Units	P1	P2	P3	P4
1	3.5	4.0	3.5	12.0
2	4.3	5.5	7.5	3.5
3	8.7	3.5	6.0	8.0

There is no storage facility is available between the processing units which means that unit 'k' has to hold a product that it has processed until unit 'k+1' becomes free. However, products that have been processed by the last unit are immediately sent to the storage unit.

A unit can begin processing a product immediately after it has finished processing the previous product and has sent it to the next unit. Also, the time required to transfer products from one unit to the next is negligible compared to the processing times.

The units are ready to begin processing at time zero and the production of any product can begin at any time. The objective is to find a sequence of producing the four products in order to minimize the makespan.

For the MILP formulation, the total number of products is $N = 4$, and the number of processing units is M=3 with binary variables (y_{ij}) and continuous variables (C_{ik}).

The solution of this problem has been obtained using the GAMS program from the CD with the Case Studies (6) that is given in Table 6-3. GAMS compilers users manuals and related information are available on the GAMS web site, GAMS.com.

Table 6 -3. GAMS Program for the Batch Scheduling Problem, after Karimi [6].

```
$TITLE Multiproduct Batch Plant Scheduling
* Define product and unit index sets
SETS PI Product batches to be produced /p1*p4/
    UK Four batch processing units in the plant /u1*u3/
     J Slots for products in the sequence /1*4/;
ALIAS (I, J);
* Define and initialize problem data
TABLE T(PI,UK) Processing times of products on unit UK in hours
           u1   u2  u3
    p1     3.5  4.3  8.7
    p2     4.0  5.5  3.5
    p3     3.5  7.5  6.0
    p4     12.0 3.5  8.0
PARAMETER TMIN(UK) Minimum of the processing times of products on UK;
        TMIN(UK) = SMIN(PI, T(PI,UK));
PARAMETER TP(PI,UK) Processing times of products above TMIN on UK;
        TP(PI,UK) = T(PI,UK) - TMIN(UK);
SCALAR N Number of products to be produced
      M Number of units in the plant;
      N = CARD(PI);
      M = CARD(UK);
* Define optimization variables
VARIABLES X(PI,J) Product PI is in sequence slot J
        C(I,UK) Completion time of the product in sequence
              slot I on unit UK
        MSPAN Makespan or total time to produce all products;
POSITIVE VARIABLES C;
BINARY VARIABLES X;
* Define constraints and objective function
EQUATIONS OBJFUN Minimize makespan
        ONEPRODUCT(J) Only one product should be in each slot
        ONESLOT(PI) Only one slot should be assigned to each product
        CEQ1(I,UK) Completion time recurrence 9-20c
        CEQ2(I,UK) Completion time recurrence 9-20b
        CEQ3(I,UK) Completion Time recurrence 9-20d;
OBJFUN.. MSPAN =E= SUM((I,UK) $(ORD(I) EQ N AND ORD(UK) EQ M), C(I,UK));
ONEPRODUCT(J).. SUM(PI, X(PI,J)) =E= 1;
ONESLOT(PI)..   SUM(J, X(PI,J)) =E= 1;
CEQ1(I,"u1")..  C(I,"u1") =G= C(I-1,"u1") $(ORD(I) GT 1) +
                  TMIN("u1") + SUM(PI, TP(PI,"u1")*X(PI,I));
CEQ2(I,UK) $(ORD(UK) GT 1)..
   C(I,UK) =G= C(I,UK-1) + TMIN(UK) + SUM(PI, TP(PI,UK)*X(PI,I));
CEQ3(I,UK) $(ORD(I) GT 1 AND ORD(UK) LT M).. C(I,UK) =G= C(I-1,UK+1);
* Define model and solve
 MODEL SCHEDULE /ALL/;
 SOLVE SCHEDULE USING MIP MINIMIZING MSPAN;
DISPLAY X.L, C.L, MSPAN.L;
```

The first command in the GAMS program in Table 6-3 (also called the directive) is TITLE that causes every page of the output solution to contain the title that has been specified with this directive. For this example, the title is "Multiproduct Batch Plant Scheduling" and this would appear on each page of the output solution. This directive is preceded by the '$' sign and hence these are called the Dollar Control Directives. Such directives are put in the input file to control the appearance and amount of detail in the output produced by the GAMS compiler. Also, any text that follows the asterisk '*' is treated as a comment by the compiler and hence ignored. The entire problem follows the GAMS model, the basic components of which are explained below.

SETS: These form the basic building block of a GAMS model and they correspond to the indices in the algebraic representation of models. In the example problem, PI, UK and J are the indices for the product batches, processing units and slots for products respectively. The values for these sets are enclosed within the slashes. For example, for the SET J, the number of slots is four and hence the number within the slashes are /1*4/ which is the concise way of writing, instead of writing /1, 2, 3, 4/. The next statement, ALIAS, is used to give another name to a previously declared set.

DATA: The next component of the GAMS model is DATA that consists of TABLES,

PARAMETERS AND SCALARS: In this component, all the input data is entered. Table 2 is entered in the TABLE section with the name T(PI,UK). In the PARAMETERS section, T(PI,UK) = TMIN(UK) + [T(PI,UK) - TMIN(UK)] where TMIN(UK) is the minimum processing times of products on unit UK. This not only increases the sparsity of the formulation, but also reduces the coefficients of the binary variables, thereby making the problem easier to solve (6). The SCALAR statement is used for variables that can have only single values. The function CARD() returns an integer value which corresponds to the number of elements in the set. The statement N = CARD(PI) assigns the value 4 to N.

VARIABLES: This component consists of all the decision variables of the GAMS model. Once the variables are declared, they must be assigned the type i.e. either POSITIVE, NEGATIVE, INTEGER, BINARY or FREE. Here, 'C' is a positive variable and 'X' is a binary variable. The variable that represents the quantity to be optimized must be a scalar and must be of the FREE type which means that the range of the variable is from -∞ to +∞.

EQUATIONS: This component of the GAMS model declares and defines all the equations of the problem. All equations are first declared and then defined in separate statements. GAMS has several notations to simplify complex equations. One of them is the summation notation which has two arguments: SUM(index of summation, summand). The command ORD() gives the position of an element in the set. The Dollar '$' operator is used for introducing specific conditions in the equations. For example, X$(Y EQ 5) = 8 implies that the value 8 is assigned to X only if Y is equal to the value 5. Such notations and commands are helpful to greatly simplify equations that are complex.

MODEL: This statement means that it is a group of equations. The format of this statement is the keyword MODEL followed by the model's name, followed by the list of equation names to be considered and enclosed in slashes. If all equations are to be considered for the solution, then "/ALL/" can be entered to represent the entire list of equations.

SOLVE: This statement is used to solve the model. The format sequence of the SOLVE statement is as follows:

1. The keyword "SOLVE".
2. Model name.
3. The keyword "USING".
4. The solution procedure available, like "LP", "NLP", ”MIP", etc.,
5. The keyword "MAXIMIZING" or "MINIMIZING".
6. The name of the variable to be optimized.

DISPLAY: This final statement is used to display values of specific variables at the output.

A section of the output solution obtained by solving the input program on GAMS is shown in Table 4. It gives the summary of the solution process. The minimum makespan obtained, which is given by the objective value is 34.8 hours.

Table 6-4. GAMS Output for Optimal Solution to the Batch Plant Scheduling Problem from Karimi[6].

```
Multiproduct Batch Plant Scheduling
Solution Report     SOLVE SCHEDULE USING MIP FROM LINE 53
            S O L V E     S U M M A R Y
    MODEL   SCHEDULE        OBJECTIVE  MSPAN
    TYPE    MIP             DIRECTION  MINIMIZE
    SOLVER  ZOOM            FROM LINE  53
**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL
**** OBJECTIVE VALUE              34.8000
 RESOURCE USAGE, LIMIT           2.090    1000.000
 ITERATION COUNT, LIMIT       164         1000
```

```
**** REPORT SUMMARY :       0  NONOPT
                            0  INFEASIBLE
                            0  UNBOUNDED

----    55 VARIABLE  X.L         Product PI is in sequence slot J
          1          2          3         4
P1      1.000
P2                                    1.000
P3                 1.000
P4                            1.000
----    55 VARIABLE  C.L         Completion time of the product in sequence
         U1          U2          U3
1       3.500       7.800      16.500
2       7.800      16.500      23.300
3      19.800      23.300      31.300
4      23.800      31.300      34.800
----    55 VARIABLE  MSPAN.L    =   34.800 Makespan or total time to produce all products
```

The final values of the binary variables are listed in the form of a table that shows the job sequence. Here, y_{11}, y_{24}, X_{32}, and X_{43} have a value '1'. This means that product 1 has been allotted slot 1, product 2 has been allotted slot 4, product 3 has been allotted slot 2 and product 4 has been allotted slot 3. Thus, the final sequence in which the products will be produced to minimize the makespan is P1-P3-P4-P2.

Next, the completion time for each process in each unit is listed. C_{43} is the makespan and is equal to 34.8. From this table, the Gantt chart can be drawn. Finally, the value of the variable MSPAN that corresponds to the makespan is given.

Closure

In this chapter, mixed integer linear programming was described along with its special cases. First, the mathematical structure of MILP was introduced and then some perspective was given on solving integer-programming problems. The branch and bound technique for solving mixed integer problems was then described along with an algorithm to solve general integer problems using this technique. The use of this algorithm was illustrated by solving an example. Later, a binary integer problem was solved in a similar but different way, but essentially using the same technique. The purpose was to show that binary integer programming problems could be solved by an efficient and faster method. Then mixed integer programming was introduced, and a problem was solved using the branch and bound technique. Finally, the MILP approach in batch plant scheduling was explained with the help of an example, and equations were derived to formulate such problems. Finally, the chapter closed by applying these equations to construct a mathematical model of a multiproduct batch plant scheduling problem whose solution was obtained using GAMS. Both, the GAMS code and solution for the problem were discussed.

References

1.Nemhauser, G. L., A. H. G. Rinnooy Kim and M. J. Todd, *Optimization*, Elsevier Science Publications, New York (1989).

2. Ecker, J. G. and M. Kupferschmid, *Introduction of Operations Research*, Wiley, New York (1988).

3. Ravindran, A., D. T. Phillips and J. J. Solberg, *Operations Research; Principles and Practice*, Sec. Ed., Wiley, New York (1987).

4. Murtagh, B. A., *Advanced Linear Programming: Computation and Practice*, McGraw-Hill, New York (1981).

5. Hiller, F. S. and Lieberman, G. J., *Introduction to Operations Research*, McGraw-Hill, New York (1990).

6. Karimi, I. A., "Multiproduct Batch Plant Scheduling," Chemical Engineering Optimization Models with GAMS. CACHE Design Case Studies Series, Case Study No.6, Grossmann, I. E., Ed., CACHE Corporation, Austin, Texas (1991).

7. Murty, K. G., *Linear and Combinatorial Programming*, Wiley, New York (1976).

8. Mah, R. S. H., *Chemical Process Structures and Information Flows*, Buttersworth, Boston (1990).

9. IBM Mathematical Programming System Extended/370 Primer, GH19-1091-1, 2nd Ed., IBM Corp., White Plains, New York. (1979).

10. Brooke, A., Kendrik, D., and Meeraus, A., *GAMS: A User's Guide*, The Scientific Press, Redwood City, CA (1988).

11. *The Operations Research Problem Solver,* Research and Education Association, New York (1983).

12. McMillan, Claude Jr., *Mathematical Programming: An Introduction to the Design and Application of Optimal Decision Machines*, Wiley, New York (1970).

13. Harley, R., *Linear and Nonlinear Programming*, Wiley, p. 162f, (1985).

14. Ku, H-M and I. A. Karimi, 1988, "Scheduling in Serial Multiproduct Batch Processes with Finite Interstage Storage: A Mixed Integer Linear Program Formulation," *Industrial Engineering Chemistry Research*, Vol. 27, p.1840

15. Ku, H-M and I. A. Karimi, 1990, "Completion Time Algorithm for Serial Multiproduct Batch Processes with Shared Storage, *Computers and Chemical Engineering*, Vol. 14, No. 1, p. 49

16. Floudas, C. A. and X. Lin, 2004, "Continuous-time versus Discrete-Time Approaches for Scheduling of Chemical Processes: a Review," *Computers and Chemical Engineering*, Vol. 28, p. 2109

17. Gass, S. I., *Linear Programming: Methods and Applications*, 5th Ed., McGraw-Hill, New York, (1985).

18. Wolsey, L. A. *Integer Programming*, Wiley, p. 95, (1998).

19. Floudas, C. A., *Nonlinear and Mixed-Integer Optimization*, p. 235f, Oxford University Press, Oxford, England (1995)

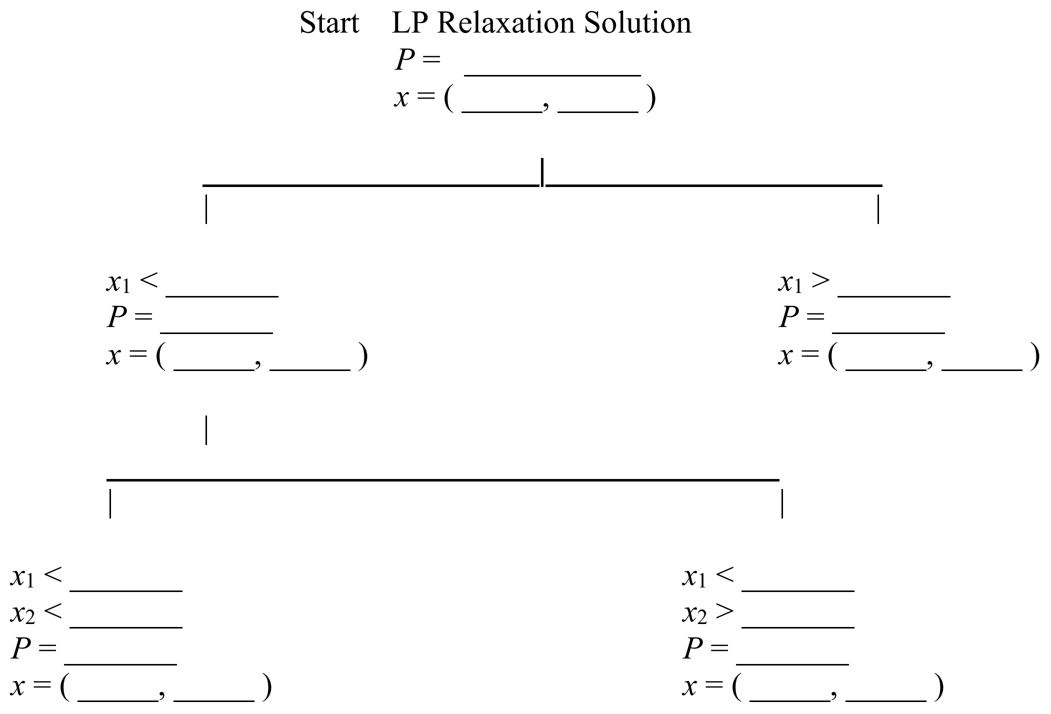
Problems

6-1. During the maximization of the following integer programming problem after S. I. Gass (17), the following subsets were obtained.

$$\begin{aligned} \text{maximize:} \quad & P(x) = 2x_1 + 5x_2 \\ \text{subject to:} \quad & 2x_1 - x_2 \leq 9 \\ & 2x_1 + 8x_2 \leq 31 \\ & x_j \geq 0 \text{ and integer} \end{aligned}$$

Subsets (not in order)

$x_1 \leq 5, x_2 \leq 2$	$x = (5, 2)$	$P = 20$
$x_1 \leq 5, x_2 \geq 0$	$x = (5, 2.625)$	$P = 23.125$
$x_1 \geq 0, x_2 \geq 0$	$x = (5.722, 2.44)$	$P = 23.66$
$x_1 \geq 6, x_2 \geq 0$	Infeasible	$P =$ -
$x_1 \leq 5, x_2 \geq 3$	$x = (3.5, 3)$	$P = 22$



a. Place the subset solutions on the branch and bound tree given above.
b. Write on the diagram all nodes that have been fathomed and explain why.
c. Write on the diagram the node that has not been fathomed and explain why.
d. Give the upper and lower bounds at this point in the solution.

6-2. Consider the following integer programming problem after Wolsey (18).

$$\begin{array}{rll} \text{Max:} & 4x_1 - x_2 & = P \\ \text{Subject to:} & 7x_1 - 2x_2 & < 14 \\ & 2x_1 - 2x_2 & < 3 \\ & x_2 & < 3 \\ & x_1,\ x_2 > 0 & \text{integers} \end{array}$$

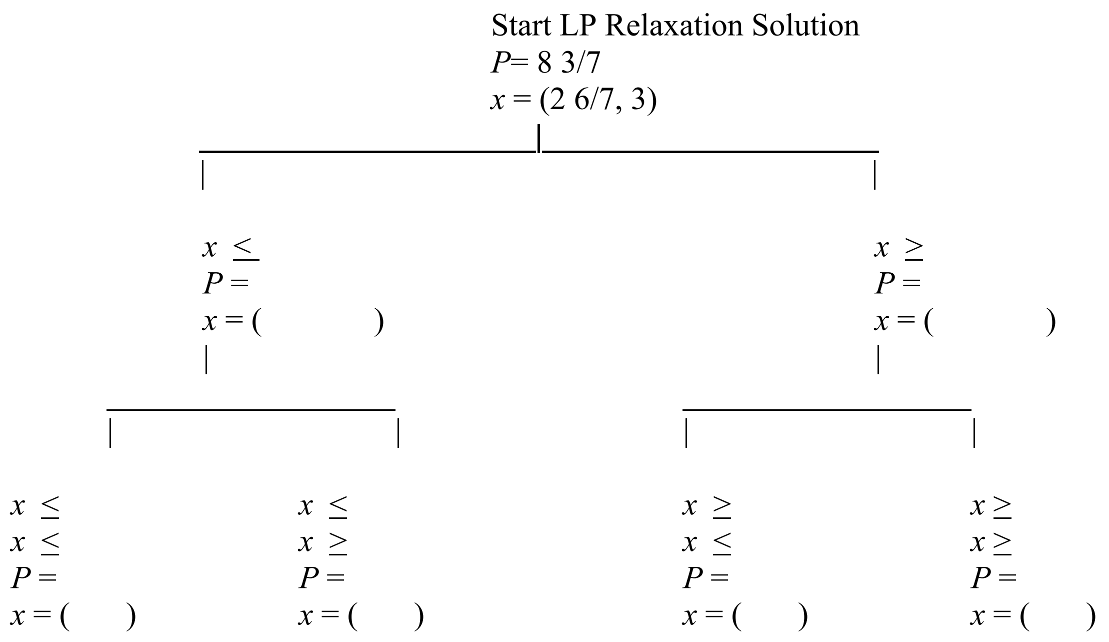
The above integer-programming problem was solved using the branch and bound algorithm. The constraints and the LP relaxation solution of all of the subproblems on the branches are listed below.

Constraints	LP Solution for Subproblems		
$x_2 \leq 3$	$x = (2\ 6/7, 3)$	$P= 8\ 3/7$	LP relaxation solution
$x_1 \leq 2 \quad x_2 \leq 3 \quad x_2 \geq 1$	$x = (2, 1)$	$P= 7$	
$x_1 \leq 2 \quad x_2 \leq 3$	$x = (2, 1/2)$	$P= 7\ 1/2$	
$x_1 \leq 2 \quad x_2 \leq 3 \quad x_2 \leq 0$	$x = (1\ ½, 0)$	$P= 6$	
$x_1 \geq 3 \quad x_2 \leq 3$	$x =$ (infeasible)	$P=$ -	

a. From the LP solutions of the subproblems given in the table below, complete the branch and bound tree. Add subscripts to the *x*'s and values used for branching to each subproblem on the attached diagram. All of the places for subproblems are not needed.

b. Write on the diagram the reasons for fathoming each subproblem.

c. Describe the procedure to locate and give the upper and lower bounds using a breadth-first strategy.



6-3. Consider the following integer programming problem after Harley (13).

$$\begin{aligned} \text{Max: } & 3x_1 + 4x_2 + 7x_3 = P \\ \text{Subject to: } & x_1 + 3x_2 + 6x_3 < 13 \\ & 2x_1 + 3x_2 + 4x_3 < 13 \\ & x_1, x_2, x_3 > 0 \text{ integer} \end{aligned}$$

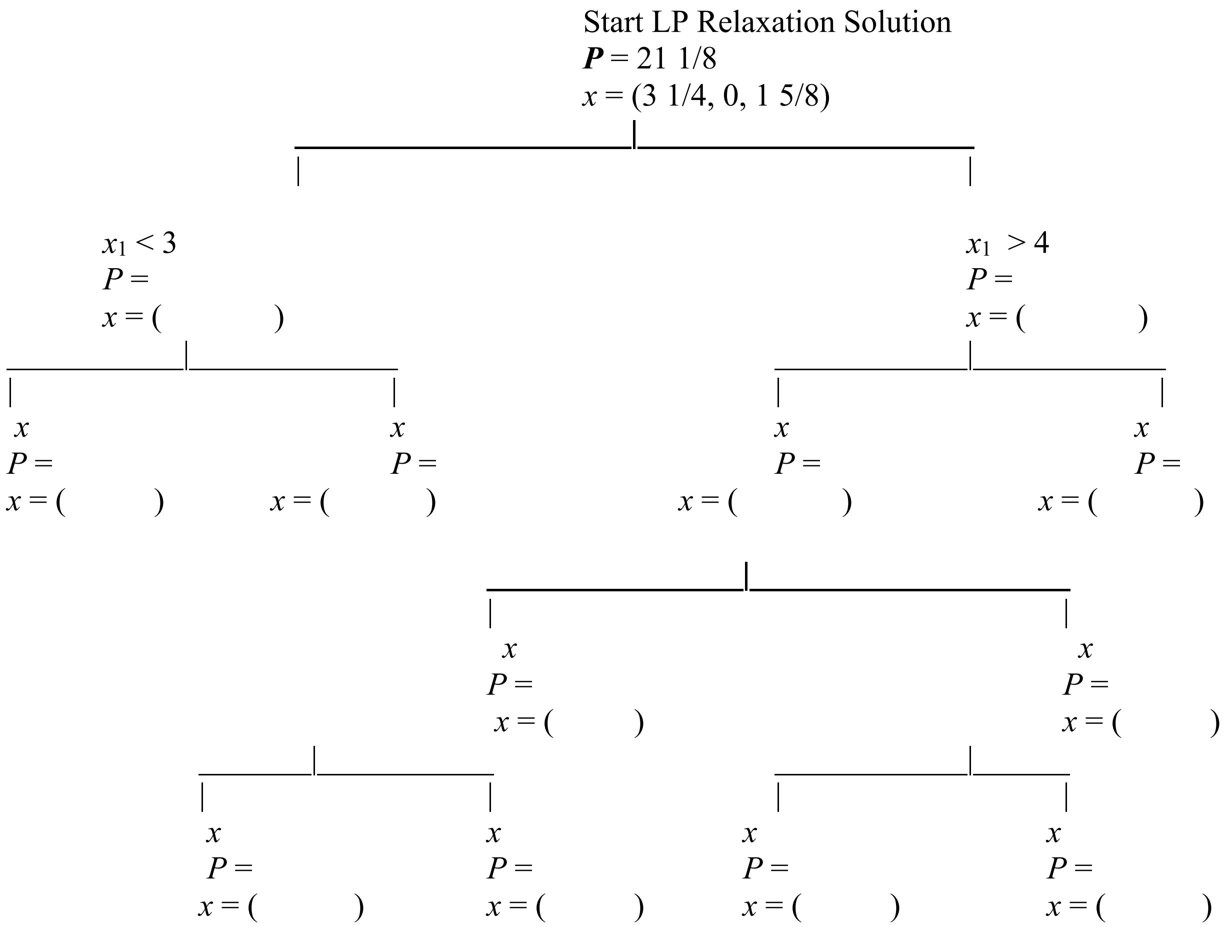
The above integer-programming problem was solved using the branch and bound algorithm. The constraints and the LP relaxation solution of all of the subproblems on the branches are listed below.

a. From the given **LP** solutions of the subproblems, complete the branch and bound tree and write the constraints added to each subproblem on the attached diagram.

b. Give the reasons for fathoming each subproblem. Write this on the diagram.

c. Describe the procedure to locate and give the upper and lower bounds at each branch using a breadth-first strategy and to locate the maximum.

Constraints			LP Solution for Subproblems		
$x_1 > 0$	$x_2 > 0$	$x_3 > 0$	$x = (31/4, 0, 15/8)$	$P = 211/8$	LP relax soln
$3 > x_1 > 0$	$x_2 > 0$	$x_3 > 0$	$x = (3, 1/3, 11/2)$	$P = 205/6$	
$3 > x_1 > 0$	$x_2 > 0$	$1 > x_3 > 0$	$x = (3, 1, 1)$	$P = 20$	
$3 > x_1 > 0$	$x_2 > 0$	$x_3 > 2$	$x = (1, 0, 2)$	$P = 17$	
$x_1 > 4$	$x_2 > 0$	$x_3 > 0$	$x = (4, 0, 11/4)$	$P = 203/4$	
$x_1 > 4$	$x_2 > 0$	$1 > x_3 > 0$	$x = (41/2, 0, 0)$	$P = 201/2$	
$x_1 > 4$	$x_2 > 0$	$x_3 > 2$	$x =$ (infeasible)	--	
$4 > x_1 > 4$	$x_2 > 0$	$1 > x_3 > 0$	$x = (4, 1/3, 1)$	$P = 201/3$	
$4 > x_1 > 4 > 0$	$x_2 > 0$	$1 > x_3 > 0$	$x = (4, 0, 1)$	$P = 19$	
$4 > x_1 > 4$	$x_2 > 1$	$1 > x_3 > 0$	$x = (4, 1, 1/2)$	$P = 191/2$	
$x_1 > 5$	$x_2 > 0$	$1 > x_3 > 0$	$x = (5, 0, 3/4)$	$P = 201/2$	
$x_1 > 5$	$x_2 > 0$	$0 > x_3 > 0$	$x = (61/2, 0, 0)$	$P = 191/2$	
$x_1 > 5$	$x_2 > 0$	$1 > x_3 > 1$	$x =$ (infeasible)	--	



	Z_U upper bound	Z_L lower bound
Start		
1st Branch		
2nd Branch		
3rd Branch		
4th Branch		

Maximum: ***P*** = $\qquad x = (\quad,\quad,\quad)$

6-4. Consider the following integer programming problem that has three binary variables, y_1, y_2, and y_3.

$$\begin{array}{ll} \text{Max:} & 3y_1 + 2y_2 + 3y_3 = \boldsymbol{P} \\ \text{Subject to:} & y_1 + y_2 + y_3 > 2 \\ & 5y_1 + 3y_2 + 4y_3 < 10 \\ & y_1, y_2, y_3 = 0,1 \end{array}$$

This problem was solved using the branch and bound algorithm. The LP relaxation solution and subproblems formed from this solution by adding constraints are given in the following table. These solutions are not in any particular order.

Constraints added	Subproblem Solutions			
for subsets	y_1	y_2	y_3	P
LP relaxation	0.6	1	1	6.8
$y_1 = 1,\ y_2 = 0$	1	0	1	6
$y_1 = 1,\ y_2 = 1\ y_3 = 0$	1	1	0	5
$y_1 = 1,\ y_2 = 1$	1	1	0.5	6.5
$y_1 = 1$	1	0.33	1	6.67
$y_1 = 1, y_2 = 1, y_3 = 1$	1	1	1	infeasible
$y_1 = 0$	0	1	1	5

a. Write the LP relaxation and subset solutions on the attached branch and bound diagram.

b. Write on the diagram the reason that each node is fathomed.

c. Give the upper and lower bounds at each level, and show that this locates the maximum.

Start LP relaxation

Z_U = ______ P = ______

Z_L = _____ y_1 = ____ y_2 = ____ y_3 = ____

Z_U = ______ y = ____ y = ____ add subscripts to y's

Z_L = _____ P = ____ P = ____

y_1 = ____ y_2 = ____ y_3 = ____ y_1 = ____ y_2 = ____ y_3 = ____

fathomed ________________

Z_U = ______ y = ____ y = ____ add subscripts to y's

Z_L = _____ y = ____ y = ____

P = ____ P = ____

y_1 = ____ y_2 = ____ y_3 = ____ y_1 = ____ y_2 = ____ y_3 = ____

fathomed _______________

Z_U = ______ y = ____ y = ____ add subscripts to y's

Z_L = _____ y = ____ y = ____

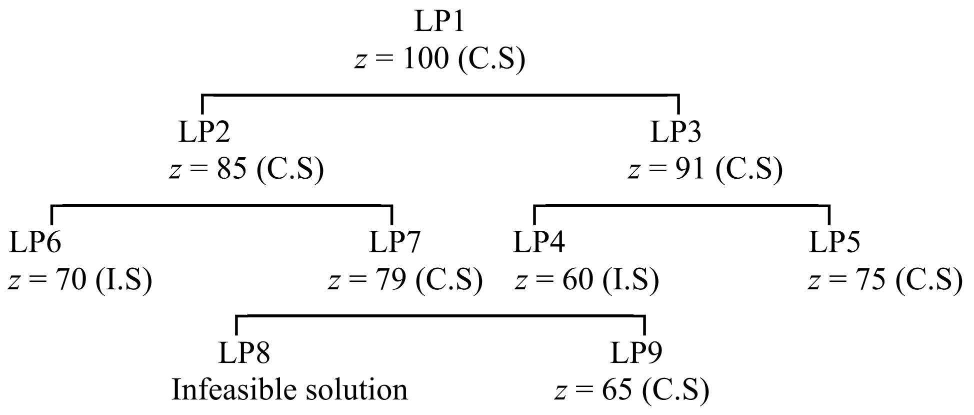
y = ____ y = ____

P = ____ P = ____

y_1 = ____ y_2 = ____ y_3 = ____ y_1 = ____ y_2 = ____ y_3 = ____

fathomed _______________ fathomed _______________

6-5.[3] During the maximization of a pure integer programming problem by the branch and bound algorithm, the following branch and bound tree is obtained at a certain stage.



Note: C.S = continuous solution, I.S = Integer solution.

a. What is the best upper bound on the maximum value of z for the integer program at this stage?
b. What is the best lower bound on the maximum value of z?
c. Indicate all the node(s) that have been fathomed and explain why.
d. Identify the node(s) that have not been fathomed and explain why not.
e. Has an optimal solution to the integer program been obtained at this stage? Explain.
f. What is the maximum absolute error on the optimal value of z if the branch and bound algorithm is terminated at this stage? What is the fractional error as a percentage of worst-case optimum?

6-6. Several integer-programming problems are given below. The branch and bound solutions are given in References 2,5, and 11.

Maximize: $z(x) = 3x_1 + 13x_2$
Subject to: $2x_1 + 9x_2 \leq 40$
$11x_1 - 8x_2 \leq 82$
x_1, x_2, non-negative integers

Maximize: $z(x) = 6x_1 + 3x_2 + x_3 + 2x_4$
Subject to: $x_1 + x_2 + x_3 + x_4 \leq 8$
$2x_1 + x_2 + 3x_3 \leq 12$
$5x_2 + x_3 + 3x_4 \leq 6$
$x_1 \leq 1, x_2 \leq 1, x_3 \leq 4, x_4 \leq 2$
x_1, x_2, x_3, x_4 non-negative integers.

Maximize: $z(x) = 10x + 20y$
Subject to: $5x + 8y \leq 60$
$x \leq 8, y \leq 4$
x, continuous variable y, non-negative integers.
The LP-relaxation of this problem is $x = 5.6$, $y = 4$ with $z = 136$.

Minimize: $z(x) = x_1 - 2x_2$
Subject to: $2x_1 + x_2 \leq 5$
$-4x_1 + 4x_2 \leq 5$
x_1, x_2, non-negative integers.

Minimize: $z(x) = 8x_1 + 15x_2$
Subject to: $10x_1 + 21x_2 \leq 156$
$2x_1 + x_2 \leq 22$
x_1, x_2, non-negative integers.

Maximize: $z(x) = -x_1 + 15x_2$
Subject to: $-x_1 + 10x_2 \leq 10$
$x_1 + x_2 \leq 6$
x_1, x_2, non-negative integers.

Maximize: $z(x) = 9x_1 + 6x_2 + 5x_3$
Subject to: $2x_1 + 3x_2 + 7x_3 \leq 35/2$
$4x_1 + 9x_3 \leq 15$
x_1, x_2, x_3, non-negative integers.

Maximize: $z(x) = 2x_1 + 3x_2 + x_3 + 2x_4$
Subject to: $5x_1 + 2x_2 + x_3 + x_4 \leq 15$
$2x_1 + 6x_2 + 10x_3 + 8x_4 \leq 60$
$x_1 + x_2 + x_3 + x_4 \leq 8$
$2x_1 + 2x_2 + 3x_3 + 3x_4 \leq 16$
$x_1 \leq 3, x_2 \leq 7, x_3 \leq 5, x_4 \leq 5$
x_1, x_2, x_3, x_4 non-negative integers.

Minimize: $z(x) = -2x_1 - 10x_2 - x3$
Subject to: $5x_1 + 2x_2 + x_3 \leq 7$
$2x_1 + x_2 + 7x_3 \leq 9$
$x_1 + 3x_2 + 2x_3 \leq 5$
$x_j = 0$ or $1, j = 1...,3$.

Minimize: $z(x) = 2x_1 + 4x_2 - 5x_3 + 7x_4$
Subject to: $x_1 + 2x_2 + 3x_3 + 3x_4 \leq 8$
$-2x_1 + 3x_2 + x_3 + 2x_4 \geq 2$
$x_j = 0$ or $1, j = 1...,4$.

Minimize: $z(x) = -2x_1 - 4x_2 - 6x_3 - 8x_4$
Subject to: $x_1 + 2x_2 - x_3 + x_4 \leq 5$
$-2x_1 + x_2 + x_3 \geq 2$
$x_j = 0$ or $1, j = 1,..4$.

Minimize: $z(x) = 2x_1 + 3x_2 - 4x_3 + 7x_4$

Subject to: $x_1 - 2x_2 + x_3 - 4x_4 \geq 1$

$x_1 - 2x_2 + 2x_3 - x_4 \leq 1$

$x_j = 0$ or $1, j = 1,..,4.$

Maximize: $z(x) = 9x_1 + 6x_2 + 5x_3$

Subject to: $2x_1 + 3x_2 + 7x_3 \leq 35/2$

$4x_1 + 9x_3 \leq 15$

$x_1 \geq 0$ and integer

Maximize: $z(x) = 4x_1 - 2x_2 + 7x_3 - x_4$

Subject to: $-x_1 + 2x_3 - 2x_4 \leq 3$

$x_1 + x_2 - x_3 \leq 1$

$6x_1 - 5x_2 \leq 0$

$x_1 + 5x_3 \leq 10$

$x_j \geq 0$ for $j = 1,..,4.$

x_j is an integer for $j = 1, 2, 3.$

6-7. (11) A hiker decides to go on a camping trip, and he does not wish to carry more than 60 pounds in his pack, but on laying out his equipment he finds its total weight to be 90 pounds. There are three objects he wants to take, so in order to decide which combination is best, he attaches a value to each so that he can take those objects which amounts to a maximum value. Suppose his data are:

Object	Value	Weight	Value/Weight
1	70	40	1.75
2	50	30	1.67
3	30	20	1.5

As seen in the data, he has listed the objects in order of decreasing value-to-weight ratio. Formulate an integer-programming model to solve this problem. What is the solution by applying the largest-ratio rule?

6-8. (12) Seventy-five hundred soldiers are to be transported across the Mediterranean sea. The army has hired the services of a shipping company that owns two types of ships. The attributes for the two ships are shown below:

	Type 1	Type 2
Capacity, in soldiers	2,000	1,000
Gallons fuel consumption/trip	12,000	7,000
Crew size, in men	250	100

Only 55,000 gallons of fuel and 900 crewmen are available. The army will pay the shipping company \$20,000 for each ship of Type 1 employed and \$10,000 for each ship of Type 2 employed.

Formulate the problem as an integer-programming problem if the objective is to maximize the revenue without violating the fuel and crew constraints? Assume that the shipping company has an ample supply of both types of ships.

Repeat the problem with the addition of the following constraints:

If any Type 2 ships are to be employed, a special cost of $2,000 is incurred, but not otherwise.

If more than two Type 2 ships are employed, an additional cost of $1000 will be incurred since some schedule changes will become necessary.

6-9. (3) It is required to produce 2000 units of a certain product on three different machines. The set-up costs, the production costs per unit, and the maximum production capacity for each machine are given below:

Machine	Set-up Cost($)	Machine Capacity	Production Cost
1	$100	600 units	$10 per unit for the first 300 units $7 per unit for the remaining 300 units
2	$500	800 units	$2 per unit for all 800 units
3	$300	1200 units	$6 per unit for the first 500 units $4 per unit for the remaining 700 units

Formulate the problem as an integer-programming problem if the objective is to minimize the total cost of producing the required lot.

6-10. (12) Three ships are to be unloaded at a certain dock in which four berths are available. The time required for unloading (in days) depends on the ship's cargoes and the unloading facilities at each berth. This data showing the days of unloading time is shown below:

Ship Berth	1	2	3
1	5	13	19
2	13	10	15
3	11	15	27
4	15	9	6

Formulate the integer problem to find the optimal assignment of ships to berths so as to minimize the total ship-days of unloading time.

6-11. (12) There are three warehouses *A*, *B*, and *C* from which supplies have to be shipped to four distributors *D*, *E*, *F*, and *G*. The various specifications are given below:

Data:

Supplies available:

A:	36
B:	28
C:	16

Distributor requirements:

D:	5
E:	10
F:	35
G:	25

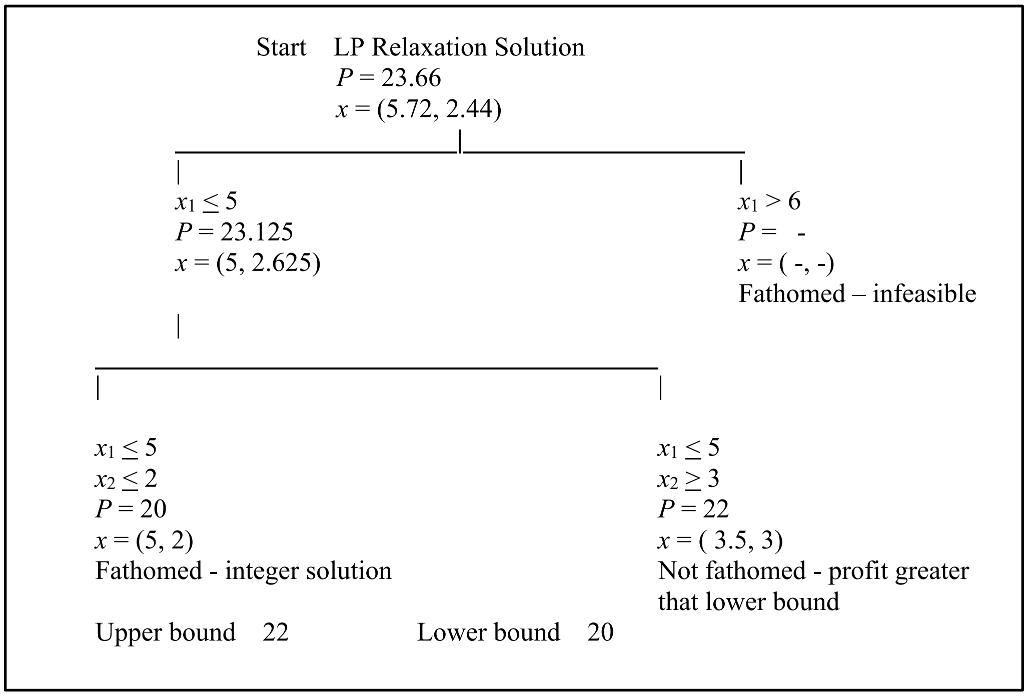
The unit shipping costs from the warehouse to the distributors:

	D	*E*	*F*	*G*
A	\$5	\$9	\$5	\$7
B	6	8	5	10
C	7	9	13	5

Formulate the problem as an integer-programming problem to find an optimal distribution that minimizes the total transportation cost, satisfies the distributor's needs and does not exceed the warehouse's supply.

Solutions to Selected Problems

6-1 Solution



6-2 Solution

a. From the given LP solutions of the subproblems, complete the branch and bound tree.

Start LP Relaxation Solution
$P = 8\ 3/7$
$x = (2\ 6/7, 3)$

$x_1 \leq 2$ $P = 7\ 1/2$ $x = (2, 1/2)$	$x_1 \geq 3$ $P =$ - $x =$ (infeasible)	Branch on x_1 $x_1 \leq 2$ $x_1 \geq 3$
$x_1 \leq 2$ $x_2 \leq 0$ $P = 6$ $x = (1½, 0)$ Less than integer soln	$x_1 \leq 2$ $x_2 \leq 1$ $P = 7$ $x = (2, 1)$ Integer soln - maximum	Branch on x_2 $x_2 \geq 1$ $x_2 \leq 0$

b. Reasons for fathoming each subproblem is on diagram.

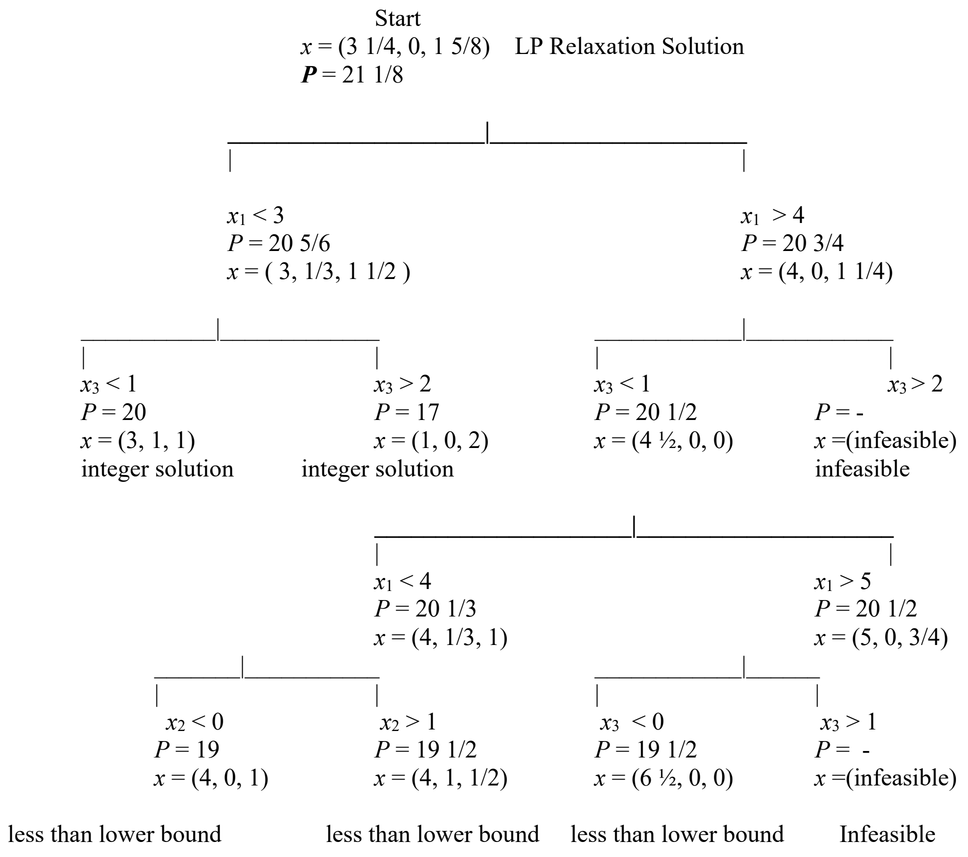
c. Describe the procedure to locate and give the upper and lower bounds using a breadth-first strategy.

	Z_u	Z_L	
Start	8 3/7	0	
1st Branch	7 1/2	0	
2nd Branch	7	7	upper bound Z_u = lower bound ZL

Maximum is: $\boldsymbol{P} = 7$, $x = (2, 1)$

The method has to proceed through two branches. At the second level the values of P for the non-integer solution, $P = 6$, is less than the incumbent solution, $P = 7$.

6-3 Solution



	Z_U	Z_L
Start	21 1/8	0
1st Branch	20 5/6	0
2nd Branch	20 1/2	20
3rd Branch	20 1/2	20
4th Branch	20	20

Maximum is: $P = 20$ $x = (3, 1, 1)$

The method has to proceed through four branches. At the third level, values of P for non-integer solutions are greater than the incumbent solution for $P = 20$.

6-4 Solution

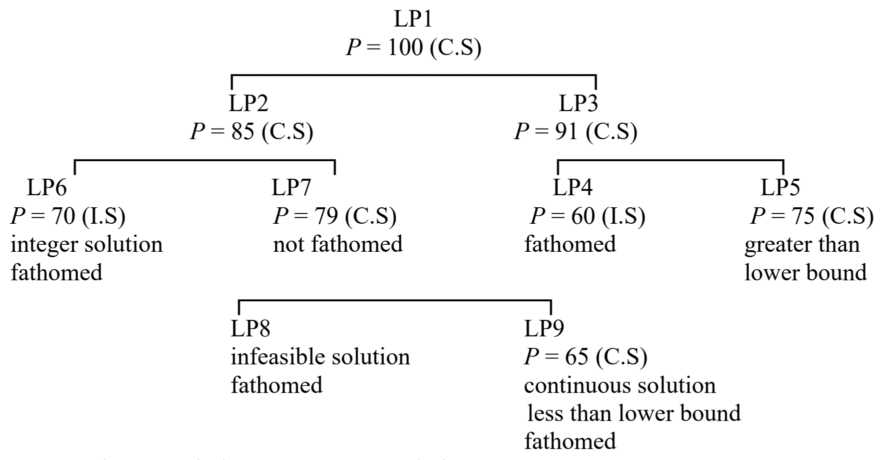
```
                                Start LP relaxation

ZU = 6.8                 P = 6.8
ZL = 0                   y1 = 0.6   y2 = 1  y3 = 1
                  ______________________|______________________
                  |                                            |
ZU = 6.67         y1 = 0                                       y1 = 1
ZL = 5            P =    5                                     P = 6.67
                  y1 = 0   y2 = 1 y3 = 1                       y1 = 1 y2 = 0.33  y3 = 1
                  fathomed: integer solution

                          _______________________________|____________
                          |                                           |
ZU = 6.5                  y1 = 1                                      y1 = 1
ZL = 6                    y2 = 0                                      y2 = 1
                          P = 6                                       P = 6.5
                          y1 = 1  y2 = 0  y3 = 1           y1 = 1  y2 = 1   y3 = 0.5
                  fathomed: integer solution
                  optimal solution
                          _______________________________|____________
                          |                                           |
ZU = 6                    y1 = 1                                      y1 = 1
ZL = 6                    y2 = 1                                      y2 = 1
                          y3 = 0                                      y3 = 1
                          P = 5                                       P =  -
                          y1 = 1   y2 = 1  y3 = 0             y1 =  -  y2 =  -   y3 =  -
                          fathomed: integer solution          fathomed: infeasible
```

6-5 Solution

During the maximization of a pure integer-programming problem by the branch and bound algorithm, the following branch and bound tree is obtained at a certain stage.



Note: C.S = continuous solution, I.S = Integer solution.

a. The upper bound on the maximum value of P for the integer program at this stage is 75.

b. The lower bound on the maximum value of P is 70.

c. Nodes fathomed are:

LP6 $P = 70$ integer solution.
LP4 $P = 60$ integer solution
LP8 $P = 60$ infeasible solution
LP9 $P = 65$ continuous solution less than lower bound.

d. Node not fathomed is LP5 $P = 75$ continuous solution greater than lower bound

e. The optimal solution to the integer program has not been obtained at this stage. The optimal solution may be larger than $P = 70$ by branching on LP5 $P = 75$.

f. The bounds on the optimal value for P at this stage, i.e., $75 > P_{\text{opt}} >$