Additional Branch and Bound Topics

The first part of this appendix extends the basic ideas of branch and bound to problems that contain both continuous and integer variables. The discussion is restricted to the case in which the integer variables are binary. In the second part of this appendix, we present the *additive algorithm* developed by Egon Balas to solve pure 0-1 integer linear programs. The interesting aspect of this algorithm is that it does not use the linear programming relaxation for the construction of the bound.

0-1 Mixed-Integer Linear Programming

The branch and bound algorithm described in Section 8.3 can be used to solve virtually all optimization problems containing integer variables, but problem classes will differ in the implementation of the subroutines *Branch*, *Approximate*, and *Variable Fixing*. We now discuss some of the issues that arise when trying to solve a 0-1 MILP. The problem has the following form

$$\text{Maximize } \mathbf{c}_1\mathbf{x} + \mathbf{c}_2\mathbf{y}$$

$$\text{subject to } \mathbf{A}_1\mathbf{x} + \mathbf{A}_2\mathbf{y} \le \mathbf{b}$$

$$x_j \ge 0,\ j = 1,\ldots,n_1$$

$$y_j = 0 \text{ or } 1,\ j = 1,\ldots,n_2$$

where $\mathbf{x}$ is an n_1-dimensional vector of real variables, $\mathbf{y}$ is an n_2-dimensional vector of binary variables, and $n = n_1 + n_2$. The number of structural constraints is m, and the parameters $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{A}_1$, $\mathbf{A}_2$ and $\mathbf{b}$ are appropriately sized arrays.

Bound

The most obvious relaxation of this problem is obtained by removing the integrality requirements on the binary variables. The result is a bounded variable linear program with the same format as above except the integrality requirements are replaced by

$$0 \le y_j \le 1,\ j = 1,\ldots,n_2$$

Assume at some point in the enumeration process that the sets S_k^+, S_k^- and S_k^0 are available. The linear programming relaxation of the problem is

$$z_{\text{UB}}^k = \text{Maximize } \mathbf{c}_1\mathbf{x} + \sum_{j \ S_k^0} c_{2j} y_j + \sum_{j \ S_k^+} c_{2j}$$

$$\text{subject to } \mathbf{A}_1\mathbf{x} + \mathbf{A}_2\mathbf{y} \le \mathbf{b}$$

$$x_j \ge 0, \ j = 1, \ldots, n_1$$

$$0 \le y_j \le 1, \ j \quad S_k^0$$

$$y_j = 1, \ j \quad S_k^+$$

$$y_j = 0, \ j \quad S_k^-$$

where the third term in the objective function is a constant, and the last two constraints fix a subset of the binary variables to either 0 or 1. The solution to this LP provides an upper bound to the set of solutions associated with node k. If the problem is infeasible, the node can be fathomed.

Note that we can no longer use Eq. (4) in Chapter 8 to reduce z_{UB}^k by its fractional part and thus obtain a more powerful bound. Because some of the variables are allowed to be noninteger, the optimal objective value may be fractional.

Approximate

Obtaining a feasible integer solution from the relaxed solution is not, in general, simple since rounding will rarely produce the desired result. Of course, if the relaxed solution has all integer values for **y**, it is also feasible to the original problem. In this case, it is considered as a replacement for the incumbent $\mathbf{y}_{\text{B}}$ in the *Update* subroutine. In any case, when the relaxed solution satisfies the integrality requirement the node is fathomed and backtracking occurs.

Example 3 (*Facility Location Problem*)

To illustrate this branch and bound procedure for a 0-1 MILP we consider a simplification of the facility location problem discussed in Section 7.4. Figure A1 gives the unit costs of transporting a commodity from each of

three proposed warehouse locations to five different customer sites. The demand for the commodity is given at the bottom of the matrix.

	Customer				
Warehouse	1	2	3	4	5
1	15	15	16	11	11
2	13	11	15	9	6
3	8	12	11	7	8
Demand	5	10	15	5	10

Figure A1. Data for facility location problem

We must decide which of the three locations should have a warehouse. The capacity of a warehouse, if built, is 30. The cost of building a warehouse at location 1, 2 or 3 is 200, 300 and 200, respectively.

We use the model that defines the variables as a proportion of demand rather than as the amount shipped. The corresponding LP relaxation has a feasible region that is closer to the MILP feasible region and provides better bounds for the fathoming test. Recall that the decision variables are

x_{ij} = fraction of jth customer's demand met from warehouse i

y_i = 1 if warehouse is built at location i; 0 otherwise

and the model is:

$$\text{Minimize } z = \sum_{i=1}^{m} \sum_{j=1}^{n} d_j c_{ij} x_{ij} + \sum_{i=1}^{m} f_i y_i$$

(All demand must be met) $\sum_{i=1}^{m} x_{ij} = 1, \qquad j = 1,\ldots,n$

(Warehouse capacity limits) $\sum_{j=1}^{n} d_j x_{ij} \le u_i y_i, \qquad i = 1,\ldots,m$

(Nonnegativity) $x_{ij} \ge 0, \qquad i = 1,\ldots,m;\ j = 1,\ldots,n$

(Integrality) $y_i = 0 \text{ or } 1, \qquad i = 1,\ldots,m$

$x_{ij} \le y_i, \qquad i = 1,\ldots,m;\ j = 1,\ldots,n$

The last set of mn constraints is redundant, but its inclusion can increase the computational efficiency of the B&B algorithm significantly.

The idea is to explicitly limit the use of any shipping link whose corresponding warehouse is not opened. The constraints force the values of the y_i variables to assume the maximum value of the amount shipped (that is, $\max_j x_{ij}$) from warehouse *i*. This produces a tighter LP feasible region and hence better bounds.

To form the linear programming relaxation, the integrality constraints are replaced by simple bounds $0 \le y_i \le 1$. The search tree resulting from the branch and bound computations is illustrated in Fig. A2. Table A1 provides the details for each iteration. The vector $\mathbf{y}_{LB}$ gives the relaxed solution for the 0-1 variables with z_{LB} the objective value. Because we are minimizing rather than maximizing, the LP returns a lower bound rather than an upper bound as was the case up until now. This means that we must reverse the sense of the inequality in condition (3) in Chapter 8. With this modification, the appropriate test for fathoming node *k* becomes

$$z_{LB}^{k} \ge z_B \tag{A1}$$

The separation variable is identified in the column labeled *s* and was chosen to be the y_i closest to 0.5 amongst all fractional y_i; that is, $y_s = \min_i\{|y_i - 0.5|\}$. When the value of the fraction was exactly 0.5, the y_s = 1 node was explored first. Because we are solving a mixed-integer program, condition (4) in Chapter 8 can no longer be use to round the LP objective value.

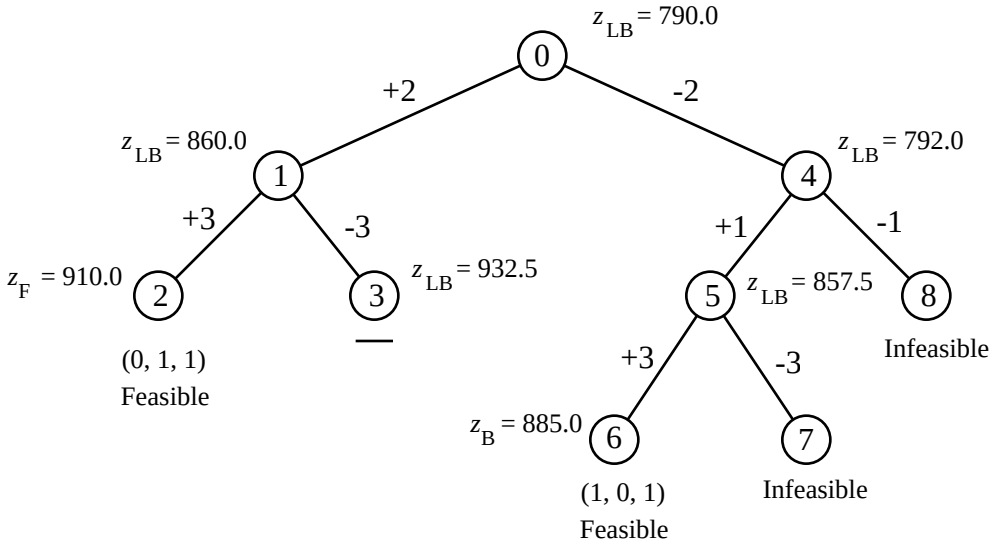

Figure A2. Search tree for warehouse location problem

The first feasible solution is uncovered at node 2 yielding an objective value $z_F = 910$. The algorithm then backtracks to node 3 where the LP objective value is $z_{LB} = 932.5$. Condition (A1) allows us to fathom this node so we backtrack to node 4. A second feasible solution is found at node 6 which turns out to be the optimum. This is confirmed after examining two more nodes. A final point about the example is that from the data we see that the total demand is 45 and the capacity of each warehouse is 30. Consequently, all solutions that specify only one warehouse can be fathomed immediately without attempting the LP relaxation. When solving integer programs, it is important to identify these types of problem-dependent restrictions. They often occasion large reductions in the computational effort.

Table A1. B&B Results for Warehouse Location Problem

Node, k	Level, l	P_k	z_{LB}	$\mathbf{y}_{LB}$	z_B	$\mathbf{y}_B$	s	Action
0	0	Ø	790.0	(0, 0.5, 1)	*M*	—	2	Set y_2=1
1	1	(+2)	860.0	(0, 1, 0.5)	*M*	—	3	Set y_3=3
2	2	(+2, +3)	910.0	(0, 1, 1)	910	(0, 1, 1)	—	Backtrack
3	2	(+2, –3)	932.5	(1, 1, 0)	910	(0, 1, 1)	—	Fathom and backtrack
4	1	(–2)	792.0	(0.5, 0,1)	910	(0, 1, 1)	1	Set y_1=1
5	2	(–2, +1)	857.5	(1, 0, 0.5)	910	(0, 1, 1)	3	Set y_3=1
6	3	(–2, +1, +3)	885.0	(1, 0, 1)	885	(1, 0, 1)	—	Backtrack
7	3	(–2, +1, –3)	Infeas.	—	885	(1, 0, 1)	—	Backtrack
8	2	(–2, –1)	Infeas.	—	885	(1, 0, 1)	—	Stop

Additive Algorithm for the Pure 0–1 Integer Programming

We now present a B&B algorithm that can be used to solve a 0-1 integer program without relying on linear programming to find upper bounds. The approach is due to Egon Balas and is referred to as the additive algorithm. We write the model as

$$\text{Maximize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad i = 1, \ldots, m$$

$$x_j = 0 \text{ or } 1, \quad j = 1,\ldots,n$$

For reasons that will soon become apparent it is assumed that all constraints are of the "less than or equal to" type. If a model is not in this form, the following transformations can be used to achieve it.

- If some constraint i is of the "greater than or equal to" type, make the following substitution.

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad \Rightarrow \quad \sum_{j=1}^{n} -a_{ij} x_j \leq -b_i$$

- If some constraint i is an equality, replace it with the following inequalities.

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \quad \Rightarrow \quad \begin{cases} \displaystyle\sum_{j=1}^{n} a_{ij} x_j \leq b_i \\ \displaystyle\sum_{j=1}^{n} -a_{ij} x_j \leq -b_i \end{cases}$$

It is also assumed that all coefficients c_j in the objective function are nonpositive. If $c_j > 0$, we replace x_j with $1 - \hat{x}_j$, where $\hat{x}_j$ is a binary variable. This transformation introduces constants on the left-hand side of the constraints that must be moved to the right. Constant terms in the objective function are ignored during the optimization but added back when the solution is found.

As an illustration of the variable transformation step, consider the knapsack example below.

$$\text{Maximize} \quad z = 5x_1 + 3x_2 + 7x_3$$

$$\text{subject to} \quad 4x_1 + 2x_2 + 5x_3 \leq 8$$

$$x_j = 0 \text{ or } 1, \quad j = 1, 2, 3$$

To obtain negative coefficients in the objective, each of the variables must be transformed. The revised problem in the form required by the algorithm is

$$\text{Maximize} \quad z = -5\hat{x}_1 - 3\hat{x}_2 - 7\hat{x}_3 + 15$$

$$\text{subject to} \quad -4\hat{x}_1 - 2\hat{x}_2 - 5\hat{x}_3 \leq 8 - 11 = -3$$

$$\hat{x}_j = 0 \text{ or } 1, \quad j = 1, 2, 3$$

To use implicit enumeration as the solution technique, it is necessary to define a relaxation that will provide bounds and allow us to test for feasibility.

Bound

A very simple relaxation of an integer program is one that requires only addition to obtain a solution. If we are at node k in the search tree with sets S_k^+, S_k^- and S_k^0, the problem under consideration can be written

$$\begin{aligned} z_{\text{UB}}^k = \text{ Maximize } & \sum_{j \ S_k^0} c_j x_j + \sum_{j \ S_k^+} c_j \\ \text{subject to } & \sum_{j \ S_k^0} a_{ij} x_j \le r_i, \quad i = 1, \ldots, m \qquad \text{(A2)} \\ & x_j = 0 \text{ or } 1, \ j \quad S_k^0 \end{aligned}$$

where the second term in the objective function is a constant -- the contribution of the variables fixed at 1. Of course, the variables fixed at 0 do not affect the objective function. To simplify the notation, we have introduced r_i to represent the right-hand side of constraint i; that is,

$$r_i = b_i - \sum_{j \ S_k^+} a_{ij}, \ i = 1, \ldots, m$$

The value r_i is the original right-hand-side value less the coefficients of the variables that are set to 1.

Because $c_j \le 0$, a relaxed solution $\mathbf{x}^k$ that maximizes the objective is obtained by setting all the free variables $x_j = 0, j \quad S_k^0$. The corresponding objective value is computed by summing the coefficients of the variables fixed to 1; that is, $z_{\text{UB}}^k = \sum_{j \ S_k^+} c_j$. In addition, if $\mathbf{r} = (r_1, \ldots, r_m) \ge 0$ implying that all the constraints are satisfied, $\mathbf{x}^k$ is optimal to the IP at node k. In this case, we update the incumbent by putting $z_{\text{B}} \leftarrow \max\{z_{\text{B}}, z_{\text{UB}}^k\}$ and backtrack.

If some of the constraints are not satisfied when all free variables are set to zero, neglecting these constraints is a relaxation of the problem.

The objective value obtained is an upper bound. The solution to this relaxation is always integer, but some of the constraints in Eq. (A2) may be violated.

For the transformed knapsack example, when all variables are free at node 0 the bound obtained is $z_{UB}^{0} = 0$ (the constant 15 will be ignored for the remainder of this section). Clearly, this is an upper bound on the optimal objective value since all the objective coefficients are negative. Given $r_1 = -3$ for the constraint, we see, however, that the solution is not feasible.

Approximate

This procedure is aimed at finding a feasible solution at a given node that can be used in subsequent fathoming tests. For the additive algorithm, we simply examine the values of r_i to see if each is positive. If so, a feasible solution has been found.

Variable Fixing

For a 0-1 integer program, various logical tests may be included to determine whether a particular node in the search tree can be fathomed because it admits no feasible solutions. Again assume that the enumerative process has progressed to node k and consider constraint i. Let t_i be the sum of the negative coefficients of the free variables. Thus

$$t_i = \sum_{j \ S_k^0} \min\{0, a_{ij}\}$$

represents the smallest possible left-hand-side value for constraint i and is always nonpositive. The right-hand-side value r_i may be positive, zero, or negative. When all the r_i are nonnegative, the solution to the relaxation is feasible and optimal for the node.

If r_i is negative for some constraint i and $t_i > r_i$, there is no feasible solution in the set represented by the node. This follows because constraint i cannot be satisfied even if all the free variables with negative structural coefficients are set to 1 and the remaining free variables set to 0. This is one feasibility test. If a node fails the test, it is fathomed and the process backtracks.

For example, consider the constraint

$$\sum_{j \in S_k^0} a_{ij}x_j = -6x_1 + 5x_2 + 2x_3 - 3x_4 \leq -12 = r_i$$

The computations give $t_i = -9 > r_i = -12$ so node k is fathomed. For the knapsack example at node 0, $r_1 = -3$ and $t_1 = -4 - 2 - 5 = -11$ so the feasibility test is satisfied. This indicates that we must continue to enumerate.

Similar reasoning can be used to determine when free variables must be fixed to 1 or 0 to assure feasibility.

- If $a_{ij} > 0, j \in S_k^0$ and $a_{ij} + t_i > r_i$, x_j must be set to 0 for feasibility.
- If $a_{ij} < 0, j \in S_k^0$ and $-a_{ij} + t_i > r_i$, x_j must be set to 1 for feasibility.

In this manner, variables may be fixed outside the usual enumeration process thus reducing the size of the search tree. For the knapsack example, these conditions do not indicate that any of the variables can be fixed.

These tests together with the simple relaxation, can be incorporated in an implicit enumeration scheme to solve the pure 0-1 IP. Because only addition is used at each step in the computations, the full procedure is called the "additive" algorithm.

Branch

The enumeration procedure requires that a separation variable be chosen at each live node. One method is to choose the variable that *most* reduces the infeasibility of the current solution. To implement this idea let

$$R_k = \{j : j \in S_k^0 \text{ and } a_{ij} < 0 \text{ for some } i \text{ such that } r_i < 0\}$$

If there is no i such that $r_i < 0$, the node is fathomed; otherwise, at least one variable whose index is an element of R_k must equal 1 in any feasible solution. Therefore, we can separate on some $x_j, j \in R_k$, and then branch to the successor node corresponding to $x_j = 1$. The following rule chooses such a $j \in R_k$ in an attempt to move toward feasibility. Define

$$I_k = \sum_{i=1}^{m} \max\{0, -r_i\}$$

to be the infeasibility of (A2). By choosing x_j for branching, the infeasibility at the successor node is

$$I_k(j) = \sum_{i=1}^{m} \max\{0, -r_i + a_{ij}\}$$

We choose x_s to minimize this value; that is,

$$I_k(s) = \min_{j \ R_k} I_k(j)$$

For example, if the constraints at node k are

$$-6x_1 - 2x_2 + 2x_3 \le -3$$

$$-3x_1 - 4x_2 + x_3 \le -2$$

$$7x_1 + 5x_2 - 5x_3 \le 4$$

then $R_k = \{1,2\}$, $I_k(1) = 3$ and $I_k(2) = 2$ so x_3 is chosen as the separation variable. Adapting, once again, the depth-first rule for branching simplifies the representation of the search tree. A consequence of this rule and of branching to $x_k = 1$ is that the path vector P_k uniquely determines the remaining enumeration required.

Example 4 (*Knapsack Problem*)

Consider the transformed knapsack problem introduced above. Table A2 describes the solution obtained with additive algorithm. At node 0, the infeasibility index $I_0 = 3$ and $R_0 = \{1, 2, 3\}$. Also, $I_0(1) = 0$, $I_0(2) = 1$ and $I_0(3) = 0$ implying that both $\hat{x}_1$ and $\hat{x}_3$ will reduce the infeasibility to 0 if either is set to 1. We arbitrarily choose $\hat{x}_3$. At node 1, the value of r_1 is positive so a feasible solution has been obtained. The algorithm thus backtracks to node 2 where the variable fixing procedure indicates that $\hat{x}_1$ should be set to 1. Branching to node 3 yields a second feasible solution with $P_3 = (\underline{-3}, \underline{+1})$. Because all the components of P_3 are underlined, the termination criterion is met and the computations cease. The optimal solution is $\hat{\mathbf{x}} = (1, 0, 0)$ with $z_B = -5$. In terms of the original problem statement we have $\mathbf{x} = (0, 1, 1)$ with $z_{IP} = 10$.

Table A2. Additive Algorithm Results for Knapsack Example

Node, k	Level, l	P_k	z_{UB}	t_1	r_1	z_B	$\hat{\mathbf{x}}_B$	s	Action
0	0	Ø	0	–11	–3	$-M$	—	3	Set $\hat{x}_3 = 1$
1	1	(+3)	–7	–6	2	–7	(0, 0, 1)	—	Backtrack
2	1	(–<u>3</u>)	0	–6	–3	–7	(0, 0, 1)	—	Fix variable $\hat{x}_1 = 1$
3	2	(–<u>3</u>, +<u>1</u>)	–5	–2	1	–5	(1, 0, 0)	—	Stop

Exercises

1. You are using Balas's additive algorithm to solve the following 0-1 IP.

$$\begin{array}{lrrrrrrrrl}
\text{Maximize} & -10x_1 & -5x_2 & -7x_3 & -2x_4 & -8x_5 & -11x_6 & -4x_7 & -12x_8 & \\
\text{subject to} & 3x_1 & -x_2 & -5x_3 & -3x_4 & +2x_5 & +2x_6 & +8x_7 & +4x_8 & \le -6 \\
& -2x_1 & +3x_2 & +1x_3 & -10x_4 & +1x_5 & -4x_6 & -6x_7 & +2x_8 & \le -9 \\
& -5x_1 & -4x_2 & -10x_3 & +2x_4 & -2x_5 & -3x_6 & -5x_7 & -3x_8 & \le -8 \\
& 3x_1 & +3x_2 & +4x_3 & -5x_4 & +5x_5 & -x_6 & +8x_7 & -4x_8 & \le -4
\end{array}$$

$$x_j = 0 \text{ or } 1, \quad \text{for } j = 1,\ldots,8$$

   a. At node 0 of the search tree, use the feasibility tests discussed at the end of Section 8.3 (variable fixing) to fix as many variables as possible. Determine the first separation variable.

   b. Say you are at an intermediate point in the enumeration process. The following information concerning the search tree is given:

$$P_k = (-5, +\underline{1}, +2, -\underline{8}) \text{ and } z_B = -99{,}999$$

   Execute the additive algorithm for as many iterations as required until backtracking is indicated. Construct a table showing all information associated with the nodes created in the search tree from the current point forward including the node reached after the backtrack operation.

2. Consider the 0-1 knapsack problem below.

$$\begin{array}{ll}
\text{Maximize} & z = 84x_1 + 48x_2 + 25x_3 + 29x_4 + 128x_5 \\
\text{subject to} & 49x_1 + 30x_2 + 19x_3 + 29x_4 + 91x_5 \le 130
\end{array}$$

$$x_j = 0 \text{ or } 1, \; j = 1,\ldots,5$$

   a. Solve with Balas's algorithm by hand showing the implicit enumeration tree that results. Do not use a computer.

   b. Solve by hand using branch and bound with LP relaxation. Construct the search tree and show all computations.

3. You are given the following 0-1 integer program.

$$\begin{aligned}
\text{Maximize } & -10x_1 - 5x_2 - 7x_3 - 2x_4 - 8x_5 - 11x_6 - 4x_7 - 12x_8 \\
\text{subject to } & 3x_1 - x_2 - 5x_3 - 3x_4 + 2x_5 + 2x_6 + 8x_7 + 4x_8 \le -6 \\
& -2x_1 + 3x_2 + x_3 - 10x_4 + x_5 - 4x_6 - 6x_7 + 2x_8 \le -9 \\
& -5x_1 - 4x_2 - 10x_3 + 2x_4 - 2x_5 - 3x_6 - 5x_7 - 3x_8 \le -8 \\
& 3x_1 + 3x_2 + 4x_3 - 5x_4 + 5x_5 - x_6 + 8x_7 - 4x_8 \le -4 \\
& x_j = 0 \text{ or } 1, \; j = 1,\ldots,8
\end{aligned}$$

a. For node 0 of the search tree, use Balas's feasibility tests to fix as many variables as possible. Determine the first separation variable.

b. Say you are at an intermediate point in the enumeration process. The following information concerning the search tree is given at node *k*.

$$S_k^+ = \{1, 2\}, \; S_k^- = \{5, 8\}$$

$$P_k = (-5, \underline{1}, 2, -\underline{8}), \; z_B = -99{,}999$$

Use Balas's algorithm to go as many steps as necessary until backtracking is indicated. Show S_k^+, S_k^-, P_k and z_B for all nodes created in the tree including the node reached after the backtrack operation.

4. For the problem

$$\begin{aligned}
\text{Maximize } & 84x_1 + 48x_2 + 25x_3 + 29x_4 + 128x_5 \\
\text{subject to } & 49x_1 + 30x_2 + 19x_3 + 29x_4 + 91x_5 \le 130 \\
& x_j = 0 \text{ or } 1, \; j = 1,\ldots,5
\end{aligned}$$

a. Solve with Balas's algorithm by hand showing the complete search tree generated and all calculations.

b. Solve by hand using branch and bound with linear programming relaxation at each node. Show the complete search tree and all computations.

5. You are given the following 0-1 integer program for which you are using the additive algorithm to solve.

$$\begin{aligned}
\text{Maximize } & -10x_1 - 30x_2 - 20x_3 - 20x_4 - 10x_5 \\
\text{subject to } & -8x_1 - 12x_2 - x_3 - 8x_4 - 2x_5 \le -16 \\
& -9x_1 - 7x_2 - 4x_3 - 10x_4 - x_5 \le -15 \\
& -6x_1 - x_2 - 8x_3 - 3x_4 - 7x_5 \le -9 \\
& x_j = 0 \text{ or } 1, \; j = 1,\ldots,5
\end{aligned}$$

At the current iteration, the solution sets are $S^- = \{2, 4\}$, $S^+ =$.

a. Draw the corresponding search tree. Starting from this point use the feasibility tests to fix as many variables as possible. Comment on the solution obtained.

b. Continue iterating and find the optimum.