

Integer Programming Methods.S1

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

$$\begin{aligned} & \text{Maximize} && \sum_{j=1}^n c_j x_j \\ & \text{subject to} && \sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, \dots, m \\ & && x_j = 0 \text{ or } 1, \quad j = 1, \dots, n \end{aligned}$$

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.

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

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

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

$$\sum_{j=1}^n a_{ij} x_j = b_i \Rightarrow \begin{aligned} & \sum_{j=1}^n a_{ij} x_j \leq b_i \\ & \sum_{j=1}^n -a_{ij} x_j \leq -b_i \end{aligned}$$

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.

$$\begin{aligned} \text{Maximize } z &= 5x_1 + 3x_2 + 7x_3 \\ \text{subject to } & 4x_1 + 2x_2 + 5x_3 \leq 8 \\ & x_j = 0 \text{ or } 1, \quad j = 1, 2, 3 \end{aligned}$$

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

$$\begin{aligned} \text{Maximize } z &= -5\hat{x}_1 - 3\hat{x}_2 - 7\hat{x}_3 + 15 \\ \text{subject to } & -4\hat{x}_1 - 2\hat{x}_2 - 5\hat{x}_3 \leq 8 - 11 = -3 \\ & \hat{x}_j = 0 \text{ or } 1, \quad j = 1, 2, 3 \end{aligned}$$

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

$$\begin{aligned} z_{\text{UB}}^k &= \text{Maximize} && \sum_{j \in S_k^0} c_j x_j + \sum_{j \in S_k^+} c_j \\ \text{subject to} &&& \sum_{j \in S_k^0} a_{ij} x_j \leq r_i, \quad i = 1, \dots, m \end{aligned} \quad (6)$$

$$x_j = 0 \text{ or } 1, \quad j \in S_k^0$$

where the second term in the objective function is a constant. The constant is the contribution of the variables set to 1. Of course the variables set to zero 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,

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

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

Because $c_j \geq 0$, a relaxed solution \mathbf{x}^k that maximizes the objective is obtained by setting all the free variables $x_j = 0, j \in S_k^0$. The corresponding objective value is computed by summing the coefficients of the variables fixed to 1; that is, $z_{UB}^k = \sum_{j \in S_k^+} c_j$. In addition, if $\mathbf{r} = (r_1, \dots, r_m) \geq 0$ 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 $z_B = \max\{z_B, z_{UB}^k\}$ 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 (6) 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

$$t_i = \min_{j \in S_k^0} \{0, a_{ij}\}$$

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 optimum 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

$$a_{ij}x_j = -6x_1 + 5x_2 + 2x_3 - 3x_4 \quad -12 = r_i$$

$$j \in S_k^0$$

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

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

to be the infeasibility of (6). By choosing x_j for branching, the infeasibility at the successor node is

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

We choose x_s to minimize this value; that is,

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

For example, if the constraints at node k are

$$-6x_1 - 2x_2 + 2x_3 = -3$$

$$-3x_1 - 4x_2 + x_3 = -2$$

$$7x_1 + 5x_2 - 5x_3 = 4$$

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 9 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 9. 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	\emptyset	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