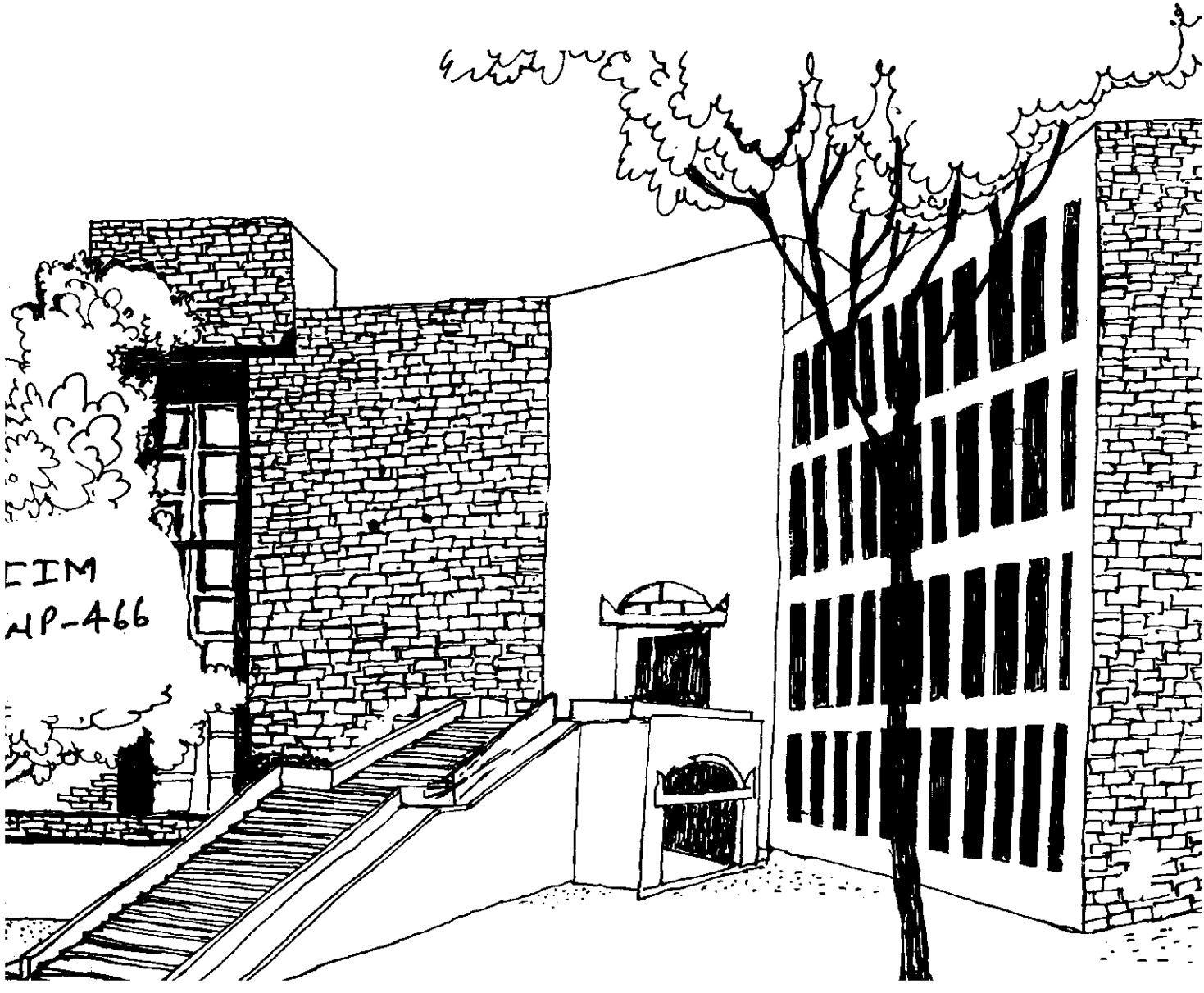
विद्याविनियोगाद्विकासः
IIM
AHMEDABAD

Working Paper



THE AGGREGATE AND BRANCH METHOD FOR SOLVING MULTI CONSTRAINT LINEAR PROGRAMS WITH ZERO-ONE VARIABLES

By

S.K. Barua

W P No. 466

July, 1983

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD-380015
INDIA

THE AGGREGATE AND BRANCH METHOD FOR SOLVING MULTI CONSTRAINT LINEAR PROGRAMS WITH ZERO-ONE VARIABLES*

S.K. Barua

Introduction

Many important class of economic problems find their mathematical models in linear programs with integer variables. Prominent among these are those that correspond to linear programs with variables taking only one of the values, zero or one. Scheduling, sequencing, capital budgeting, location of service centres in a region, and many other problems can be formulated as zero-one programs. In this paper, a new approach called Aggregate and Branch Method (ABM) for solving multi-constraint linear programs with zero-one variables has been described.

Several methods have been proposed for reducing an integer linear programming problem with bounds on the variables to an equivalent single constraint problem. The pioneering work in this area was done by Mathews in 1896. Interest in the topic revived in the seventies as reflected in the works of Anthonisse(1), Bradley (2), Padberg (6), Glover and Woolsey (3), Kendall and Zionts (5). The motivation of these researches has been to use efficient algorithms available for knapsack problem to solve the original multi constraint problem.

The motivation for the present work is the same as described above. It addresses though to a special class of integer programming problems in which variables can take a value of zero or one. The algorithm proposed in the paper first achieves an equivalent single constraint representation of the original multi-constraint problem and then uses a modified branch and search routine to solve the resulting knapsack problem.

---

*Funded by IIM, Ahmedabad

The algorithm

The original multi-constraint problem is:

$$\text{Max} \quad \sum_{j=1}^{n} c_j x_j \qquad \ldots (1)$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \text{ for } i = 1, \ldots, m$$

$$x_j = 0 \text{ or } 1$$

$a_{ij} \geq 0$, $b_i > 0$, are integers for all i and j.

The algorithm is best explained through the following steps:

Step 1: All the inequalities are converted into equations by adding slack variables.

$$\text{Max} \quad \sum_{j=1}^{n} c_j x_j \qquad \ldots (2)$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j + s_i = b_i \text{ for } i = 1, \ldots, m$$

$$x_j = 0 \text{ or } 1$$

$s_i \geq 0$ are integers

Step 2: By repeated application of any of the methods suggested for aggregating two diophantine equations, the system of equations is converted into an equivalent equation.

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

$$\text{s.t.} \quad \sum_{j=1}^{n} g_j x_j + \sum_{i=1}^{m} k_i s_i = G \qquad \ldots (3)$$

where g_j is the aggregate coefficient of the j^{th} variable

k_i is the aggregate coefficient of the i^{th} slack variable

G is the aggregate value of the right hand side

Step 3: The equivalent single constraint problem is broken up into two parts, the Main Problem and the Subproblem.

Main Problem :

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

$$\text{s.t.} \quad \sum_{j=1}^{n} g_j x_j \leq G \qquad \ldots (4a)$$

$$x_j = 0 \text{ or } 1$$

Subproblem: The feasibility of a solution x_j^* to (4a) is determined by examining whether the following equation has a feasible solution.

$$\sum_{i=1}^{m} k_i s_i = G - \sum_{j=1}^{n} g_j x_j^* \qquad \ldots (4b)$$

$$s_i \geq 0 \text{ are integers}$$

The reason for breaking up (3) into (4a) and (4b) is that it is possible to solve a knapsack problem with inequality constraint, and test the feasibility of the solution for the original constraint set, rather than solve a knapsack problem with an equality constraint straightaway.

The branch and search method suggested by Greenberg and Hegerich(4) could be used for solving (4a). The feasibility of terminal solution obtained in (4a) would then be examined by a direct substitution in the original constraint set. Whenever a feasible terminal solution better than the current lower bound solution is reached, the solution is recorded as the new lower bound solution and the branch and search routine would continue. If the terminal solution is not feasible but has an objective function value greater than the one recorded in the lower bound solution, the method proceeds to step 4.

Step 4: At this stage there is a terminal solution to (4a) with an objective function value greater than the lower bound but which violates the original constraint set. It is possible that a sub-set of the set of variables that constitute this terminal solution, would give a better feasible solution. To find such a sub-set the following procedure will be used:

Suppose a terminal solution with p x_j equal to one has been reached. These variables are first arranged in the ascending order of their objective function coefficients.

$$x_1, x_2, \cdots, x_p$$

$$c_1 \leq c_2 \leq \cdots \leq c_p$$

These x_j need not be successive in j. In order to explain the procedure it has been assumed that they are successive. Anyway, such a sequence can always be obtained by renumbering the variables.

After the variables are arranged in the above order, the first element of the sequence is put equal to zero ($x_1 = 0$) and the resulting solution is tested for feasibility. If the solution is feasible and has an objective function value greater than the current lower bound solution, the method backtracks to step 3. In case the solution is not feasible, the procedure sets $x_2 = 0$ and $x_1 = 1$ and repeats the search for better feasible solution. The method thus tests solution sequences in the following order (the number represents the index of the variable equated to zero):

1

2

⋮

p

It is clear that the solutions are arranged in the descending order of objective function value. Whenever a better feasible solution is reached, the lower bound solution is updated.

The search for a better feasible solution continues with two variables equated to zero. The order in which sub-sets with two variables put equal to zero are examined is given below:

1	2
1	3
.	.
.	.
.	.
1	p
2	3
2	4
.	.
.	.
.	.
2	p
p	(p-1)

It is quite apparent that no solution is repeated and within each sub-group the value of the objective function is non-increasing. Whenever, the value of the objective function falls below the lower bound value, the search moves on to the first element of the next subgroup.

The search continues with larger number of variables equated to zero (similar sub-grouping is done in each case). The method backtracks to Step 3 whenever the first solution in any sub-group has a lower objective function value than the current lower bound.

The lower bound solution to (4a) after all the branches have been examined would be the optimal solution to the original problem.

Preliminary Results

Limited computational experience with nine problems is recorded in Table 1.

TABLE 1

COMPUTATIONAL EXPERIENCE

S.No.	No of variables	No.of constraints	Total no. of nodes generated	No. of nodes generated till optimal solution
1	8	2	18	7
2	9	2	8	55
3	10	2	146	104
4	10	3	149	4
5	12	3	120	47
6	10	4	150	63
7	10	7	362	132
8	10	9	354	191
9	14	8	2392	599

REFERENCES

1. J.M. Anthonisse, "A Note on Equivalent Systems of Linear Diophantine Equations", Operations Research, 17, 167-177, 1973.

2. G.H. Bradley, "Transformation of Integer Programs to Knapsack Problems", Discrete Math. 1, 29-45, 1971.

3. F. Glover and R.E. Woolsey, "Aggregating Diophantine Constraints", Operations Research, 16, 1-10, 1972.

4. H. Greenberg and R.L. Hegerich, "A Branch and Search Algorithm for the Knapsack Problem", Mn. Sc., 16, 327-32, 1970.

5. K.E. Kendall and S. Zionts, "Solving Integer Programming Problems by Aggregating Constraints", Operations Research, 25, 346-51, 1977.

6. M.W. Padberg, "Equivalent Knapsack-type Formulations of Bounded Integer Linear Programs: An Alternative Approach", Naval Research Log. Quart., 19, 699-708, 1972.