Project for Quantitative Research
in Economic Development

ECONOMIC DEVELOPMENT SERIES

The Center for International Affairs

HARVARD UNIVERSITY

Project for Quantitative Research
in Economic Development

A Branch and Bound Algorithm for Zero-One Mixed Integer Programming Problems

by

Ronald E. Davis
David A. Kendrick
and
Martin Weitzman

ECONOMIC DEVELOPMENT SERIES

The Center for International Affairs
HARVARD UNIVERSITY

A Branch and Bound Algorithm for Zero-One Mixed Integer Programming Problems

by

Ronald E. Davis
David A. Kendrick
and
Martin Weitzman

Development Economic Reports, No. 69

October 1967

PROJECT FOR QUANTITATIVE RESEARCH
IN ECONOMIC DEVELOPMENT
Center for International Affairs,
Harvard University
Cambridge, Mass.

1. INTRODUCTION

Many important practical problems of optimization in management, economics, and engineering can be posed as so-called "zero-one mixed integer problems," i.e., as linear programming problems in which a subset of the variables are constrained to take on only the values zero or one. When indivisibilities, economies of scale, or combinatoric constraints are present, formulation in the mixed integer mode seems natural. Such problems arise frequently in the contexts of industrial scheduling, investment planning, and regional location, but they are by no means limited to these areas.

Unfortunately, at the present time the performance of most comprehensive algorithms on this class of problems has been disappointing. This study was undertaken in hopes of devising a more satisfactory approach. In this effort we have drawn on the computational experience of, and the concepts employed in, the Land and Doig (1960), Healy (1964) and Driebeek (1966) algorithms.[2/]

Our approach is basically a branch and bound method of enumeration. While this is not generally the most "glamorous" type of an algorithm, our experience indicates that it works well

---

1/ This research has been supported in part by the National Science Foundation under Grant GS1415 and in part by the Agency for International Development under a grant to the Project for Quantitative Research in Economic Development of Harvard University.

2/ The Land and Doig algorithm was programmed by Paul Roberts and Bob Burns of Harvard University. The Healy algorithm was programmed by two of the authors, Weitzman and Kendrick. The algorithm described here was coded by Davis. See Davis (1967).

in solving practical problems. As with any branch and bound algorithm, efficiency derives primarily from choosing branches and bounds in a manner which is computationally effective.

We proceed with a discussion of the algorithm, followed by a presentation of our computational results.

2. DESCRIPTION OF THE ALGORITHM

2.1 General Theory

Let

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

be an n-vector of "zero-one" variables, each of which is constrained to be zero or one. Let

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix}$$

be an m-vector of continuous non-negative variables. The standard form of the zero-one mixed integer programming problem is the following:

(1) minimize $z = \phi(y,x) = cy + dx$

subject to

(2) $$A\begin{bmatrix} y \\ x \end{bmatrix} \geq b \text{ for } \begin{bmatrix} y \\ x \end{bmatrix} \geq 0$$

(3) $$y_1 \leq 1 \text{ for } i = 1,\ldots,n$$

(4) $$y_1 = 0 \text{ or } 1 \text{ for } i = 1,\ldots,n$$

There are 2^n possible y vectors satisfying (4). Some of these may admit of no x satisfying (2) and hence are infeasible.

Consider the solution $(\bar{y},\bar{x})$ to the problem (1), (2), (3). (We call $(\bar{y},\bar{x})$ the solution to the "unconstrained" problem because the integerity constraints (4) are missing.) If $\bar{y}$ is already a vector of zeros and ones, the problem is solved. Far more likely, it is not. In that case, we seek more information about the local properties of the unconstrained solution.

Consider the variable y_1. Let $z_1(\tilde{y}_1)$ be the optimal value of the objective function defined parametrically in terms of $\tilde{y}_1$ for the problem (1), (2), (3) with the additional constraint $y_1 = \tilde{y}_1$, for $\tilde{y}_1 \epsilon [0,1]$. It can be shown that $z_1(\tilde{y}_1)$ is a piecewise linear convex function of $\tilde{y}_1$. (Cf. Dantzig (1963), p. 168.)

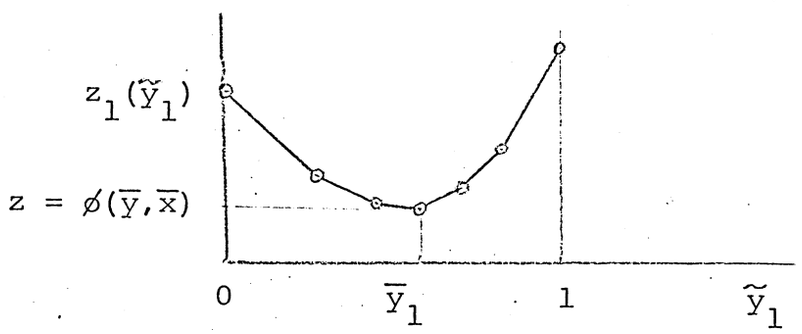


Figure 1

To map out the entire function $z_1(\tilde{y}_1)$ would require, in general, the solution of many linear programming problems. We settle instead for the more easily attainable behavior of the two linear segments to the direct right and left of $\bar{y}_1$. The slopes of these two segments can be read out of the linear programming tableau.[1/]

[1/] Strictly speaking, the assertion is true only if the basis is non-degenerate. If degeneracy is involved, we may be able to obtain only a lower bound on the slope of each segment, difficult to picture geometrically, but otherwise the results are not altered.

The slopes are obtained as follows. Suppose, first of all, that y_1 is in the optimal basis of the unconstrained problem and that $0 < \bar{y} < 1$. Let $w_1,\ldots,w_s$ be the nonbasic variables. Then in the updated simplex tableau a relation of the form

$$y_1 + \sum_{i=1}^{s} b_i w_i = \bar{y}_1$$

obtains. If $b_i > 0$, bringing w_i into the basis will decrease y_1. If $b_i < 0$, bringing w_i into the basis will increase y_1. If $b_i = 0$, bringing w_i into the basis will have no effect on y_1.

If $z + \sum_{i=1}^{s} c_i w_i = \phi(\bar{y}, \bar{x})$ is the reduced cost expression ($c_i \geq 0$ $i=1,\ldots,s$), then the "cost gradient m_0 of y_1 toward zero" (the absolute value of the left-hand slope) is

$$m_o = \min_{\substack{i=1,\ldots,s \\ b_i > 0}} \left(\frac{c_i}{b_i}\right).$$

If $b_i \leq 0$ for all $i = 1,\ldots,s$, m_0 is defined to be infinite. The "cost gradient m_1 of y_1 toward one is $m_1 = \min_{\substack{i=1,\ldots,s \\ b_i < 0}} \left(\frac{c_i}{-b_i}\right)$, or is infinite if all $b_i \geq 0$.[1/]

If y_1 is not a basis variable, $\bar{y}_1 = 0$, and m_1 is simply the reduced cost of y_1.

The cost gradients of y_1 provide an estimate of what would happen to total costs if y_1 were forced to zero or one. From the convexity of $z_1(\tilde{y}_1)$ it follows that

(5) $$z_1(1) - z_1(\bar{y}) \geq (1-\bar{y}_1)m_1 = p_1(1).$$

[1/] Cf. Healy (1964), p. 125 and p. 129. Again, m_0 and m_1 are lower bounds on the gradient in case of degeneracy.

Here m_1 is the cost gradient of y_1 toward one and $(1-\bar{y}_1)m_1$, which we denote as $p_1(1)$, is the "one penalty" of y_1, i.e., a lower bound on the increment in the objective function if y_1 were forced to one.[1/] Similarly,

$$(6) \qquad z_1(0) - z_1(\bar{y}) \geq \bar{y}_1 m_0 = p_0(1).$$

The zero and one penalties can be computed for $y_2, y_3, \ldots, y_n$ in an analogous fashion. These penalties turned out to be the prime movers in the algorithm as it evolved. They are used extensively and repeatedly for branching decisions and for evaluating bounds. It has been our experience that they perform effectively in these roles. In terms of computation time, obtaining the zero and one penalties is practically costless, since they are determined entirely from elements of the simplex tableau (which is automatically available as a result of having just employed the simplex algorithm to achieve a linear programming optimum).

2.2 Branch and Bound Search

The idea of structuring a search for an optimal mixed integer solution to equations (1)-(4) as a sequential decision problem represented by a tree in which the decision at each node is whether to set some variables to zero or one is not new (see Land and Doig (1960), Little et al. (1963), Balinski (1965), Lawler and Wood (1966)). Because it is being assumed that the reader has had at least some exposure to branch and bound procedures, we can be succinct here.

Let I be the set of all integer variables. In our procedure a node is simply a list of those integer variagles committed to zero (the set I_o), those integer variables committed to one

---

[1/] Driebeek (1966) discusses the calculation and use of the zero and one penalties in his article, p. 580.

(the set I_1), and those thus far uncommitted (the set $I - (I_o \cup I_1)$). A branch is a node that has not yet been examined in detail, but which is a potential candidate for detailed examination and about which some preliminary information is known. Only branches are stored in the computer. When a branch is evaluated it becomes a node, giving rise to the death of some old branches (including the one leading to this node) and the birth of some new ones.

A branch and bound search is defined by specifying a rule for determining, at each decision node just examined:

(1) Which new branches to create and which old ones to destroy.

(2) How to calculate relevant information, including lower bounds on the objective function for each new branch.

(3) Which branch to evaluate next.

Along with each branch is stored node specification, basis information, and conditional lower bounds on the objective function if as yet uncommitted individual integer variables were to be set to zero or one.

Let $q_o(i)$ and $q_1(i)$ denote the minimum value of the objective function if variable i were committed, respectively, to zero and one. It is understood implicitly that $q_o(i)$ and $q_1(i)$ are functions also of the node (I_o, I_1) under examination.

The efficiency of the search, that is, the number of nodes in the tree which must be examined to find and verify an optimal solution, depends upon the rules for ordering and bounding; the efficiency of the algorithm as a whole depends on the efficiency of the search and on the amount of calculation required to evaluate each node. Experimentation with various rules, based primarily on the penalty information contained in the updated simplex tableau as described in Section 2.1, has led us to adopt the procedure described below.

From any node, with current upper bound ϕ on the true minimum value of the objective function for the problem (1), (2), (3), (4).[1/]

1. Evaluate the currently specified node. That is, solve the linear programming problem (1), (2), (3) with (7) and (8) determined by the current I_o and I_1 sets. Let z' be the optimal value of the objective function for this linear program.

(7) $$\sum_{i \in I_o} y_i = 0$$

(8) $$\sum_{i \in I_1} y_i = \text{number of elements in } I$$

2. If the node is infeasible, the branch is abandoned. Go to 11.

3. If the resulting objective function equals or exceeds the current upper bound ϕ, abandon the branch. Go to 11. Otherwise continue.

4. If the solution point satisfies the integerity requirements (4), record the solution as the best yet found in the search, set the upper bound ϕ equal to the current objective function value, abandon branch and go to 11, otherwise

5. Using the zero and one cost penalties, calculate updated $q_o(i)$ and $q_1(i)$ by (9), (10) for $i \in I_u$, where I_u is the set of currently uncommitted variables ($I_u = I - I_o \cup I_1$)

(9) $$q_o(i) = \max (p_o(i) + z', q_o(i))$$

(10) $$q_1(i) = \max (p_1(i) + z', q_1(i)).$$

1/ Various methods of obtaining an initial upper bound are discussed later in the paper.

6. Calculate $\Psi(I_o, I_1) = \max_{i \in I_u} (\min(q_o(i), q_1(i)))$. $\Psi(I_o, I_1)$ is a lower bound on the objective function obtained on any branch continuing from the node (I_o, I_1).

7. If $\Psi(I_o, I_1) \geq \emptyset$, abandon current branch and go to 11.

8. If any $q_o(i)$ or $q_1(i)$ for $i \in I_u$ exceeds $\emptyset$, modify equations (7) and/or (8) accordingly.

9. If no variables have $q_o(i)$ or $q_1(i)$ exceeding $\emptyset$, or if those that do are currently at integer levels, the next two branches to be added are determined by enlarging (7) and (8) by setting to zero and one the fractional valued variable i* with the largest accumulated zero or one penalty. If $q_o(i^*) < q_1(i^*)$, the lower bound on the branch with $y_{i^*} = 0$ is $\Psi(I_o, I_1)$ and the lower bound on the branch with $y_{i^*} = 1$ is $q_1(i^*)$. Vice versa if $q_o(i^*) \geq q_1(i^*)$. Carry forward all relevant penalty information and go to 11. Otherwise continue.

10. If some variables with $q_o(i)$ or $q_1(i)$ exceeding $\emptyset$ are fractional valued in the current solution, specify a new branch with the enlarged (I_o, I_1) and appropriate penalties and go to 1.

11. Choose the next branch to evaluate according to one of the following rules:

Option 1: If condition 2., 3., 4., or 7. has been met, choose the next branch according to option 2. Otherwise, continue along current branch by evaluating node obtained from setting variable i* to the side with the smaller of $q_o(i^*)$, $q_1(i^*)$. A lower bound on the branch not taken is max $(q_o(i^*), q_1(i^*))$.

Option 2: Choose a new node by selecting the not-yet-taken branch with the least lower bound. If no lower bound is less than $\emptyset$, the search is complete.

The notion of branching from the current node is embodied in option 1. A disadvantage of this approach is that many more nodes may be evaluated than are necessary to prove convergence. One advantage is that storage requirements are minimal. Another is that, since new upper bounds ϕ are continually being discovered, the branch and bound algorithm can work more efficiently via the short cut of enlarging I_o and I_1, described in 8. above. Thus the desirable primal property of generating ever better feasible solutions can be used.

Option 2 is the "flooding" strategy of always branching from the lowest bound. The advantage of this procedure, and it can be considerable, is that only those nodes are examined which are necessary to prove convergence. A disadvantage is that large storage capacity may be required. However, since the node and basis information are stored on random access disk, computer memory was not a limiting factor for us, and this approach was utilized most of the time.

To obtain a good initial upper bound ϕ, we experimented with a variety of techniques. One of the most successful approaches was simply to run the branch and bound algorithm with option 1 and ϕ set arbitrarily large at the very beginning. This usually resulted very quickly in an all-integer solution which had a relatively low value for the objective function. Then option 2 could be applied.

2.3 Discussion of the Algorithm

Several features of the algorithm should be noted. First, each branching decision may enable several variables to be set as a necessary consequence. Suppose, for example, that there are seven integer variables, $y_1,\ldots,y_7$, and that the node

just evaluated is given by $I_o = [y_2, y_4]$ and $I_1 = [y_1]$, or $[y_2, y_4/y_1]$ for short, that the current upper bound is $\phi' = 57.4$, and that the resulting q_o and q_1 values are as below:

	q_o	q_1
y_1	forced variable	
y_2	forced variable	
y_3	57.20	57.50
y_4	forced variable	
y_5	57.30	57.35
y_6	58.40	57.20
y_7	57.10	57.30

Since $q_1(3)$ and $q_o(6)$ both exceed ϕ', any optimal continuation from the current node must satisfy $y_3 = 0$ and $y_6 = 1$. Thus I_o and I_1 are enlarged to $I_o = [y_2, y_3, y_4]$ and $I_1 = [y_1, y_6]$ and the resulting node evaluated. If the node is infeasible, then branch $[y_2, y_4/y_1]$ has been eliminated. If the mode is feasible but the objective function value exceeds ϕ', the branch $[y_2, y_4/y_1]$ is eliminated. If $\max_{i=5,7}(\min(q_o(i), q_1(i)))$ exceeds ϕ' (which would be the case if, for example, $q_o(5) = 57.45$ and $q_1(5) = 57.5$) branch $[y_2, y_4/y_1]$ has again been eliminated. If y_5 and y_7 assume integral values, the resulting lattice point is the only possible optimal one with $y_2 = y_4 = 0$ and $y_1 = 1$. If none of the above, but $q_o(i)$ or $q_1(i)$ exceeds ϕ' for $i = 5$ or $i = 7$, then the associated variable(s) may be set accordingly. For example, if the following values result,

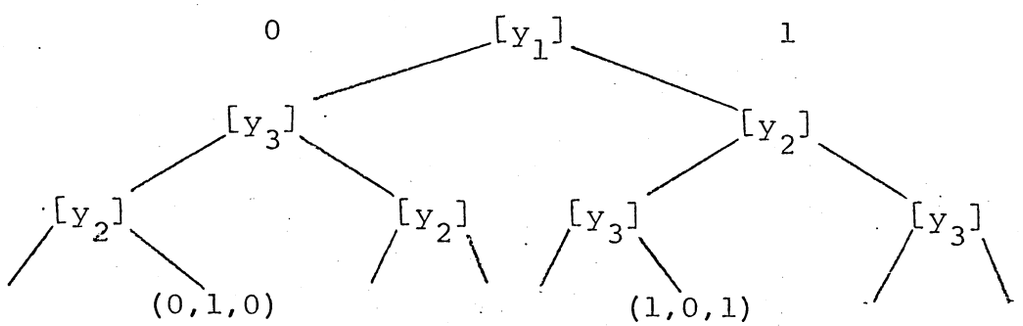
	q_o	q_1
y_5	57.80	57.38
y_7	57.36	57.32

then y_5 would have to be set to one, the resulting node evaluated and the above tests applied once again. If neither $q_o(i)$ nor

$q_1(i)$ exceeds ϕ' for $i = 5,7$ (in the above example, suppose $q_o(5) = 57.37$), then node $[y_2, y_4/y_1]$ is replaced by $[y_2, y_3, y_4/y_1, y_6]$ and the bound on continuation down the branch is max $(\min(q_o(i), q_1(i))$, or in the present case 57.37. Thus the length of the decision tree traced out by the algorithm may be considerably shorter than the number of variables in the problem.

A second feature to be noted is the behavior of the algorithm if some of the y variables are related by constraint equations. A typical case is the requirement that the sum of a set of variables is one. Forcing any of these variables to one implicitly forces each of the other variables in the set to zero. The restriction of branching decisions to noninteger valued variables avoids insertion of entirely redundant or non-binding constraints which result in no (or only degenerate) change in the optimal solution. This feature is especially useful in combinatorial problems with multiple constraints on the y variables.

Finally, the order in which the variables are set may vary from branch to branch in response to gradient information generated at various points on the surface of the feasible region. In the three variable case, for example, the decision tree might be as below:



In this way maximum bounds on branches not taken (subject to the condition that tentative decisions are made only for fractional

valued variables) are obtained.[1/]

3. EXPERIENCE WITH THE ALGORITHM[2/]

Table 1 gives a resume of our computational experience with the algorithm and comparisons with the experience of others where possible.

1/ A formal proof that such a search will produce an optimal lattice point if one exists may be found in R. E. Davis, B.A. thesis, Committee on Applied Mathematics, Harvard University, 1967.

2/ We are indebted to Jun Onaka for able programming assistance in solving the problems referred to in Table 1.

TABLE 1

Computational Experience with the Algorithm

Problem Name	Problem Number	Algorithm	Number of Integer Variables	Total Number of Variables	Total Number of Constraints	Matrix Entries	Solution Time[1/5] Minutes	Number of LP's	Number of Iterations	PIVOT STEPS
Kendrick 11 I.V.	1		11	412	116	1767				
		Healy					37.[2]	58	876	
		Driebeek					59.[3]	130	NA	
		DKW[4]					10.3	36	433	653
Markowitz -Manne	2		21	51	33	285				
		DKW					5.	62	411	
Davis 100 I.V.	3		100	317	197	1261				
		DKW					5.5	16	189	277
Balas 1	4		5	16	14	58				
		DKW					0.1	3	8	8
		Balas						3		
Balas 2	5		10	29	22	119				
		DKW					0.2	6		
		Balas						5		
Balas 3	6		9	25	19	99				
		DKW					0.1	3	11	NA
Balas 4	7		12	34	25	143				
		DKW					0.4	15	82	96
		Balas						39		
IBM Test Problem 9	8		15	68	56	253				
		DKW						128	657	805
		LIP1						69		952

(See next page for Footnotes and Notes)

Footnotes and Notes for Table 1

1/ The time on an IBM 7094 that was required to obtain the optimum mixed integer solution, including the solution to the "unconstrained" problem.

2/ After 24 minutes all but 4 of the variables could have been bounded off from an upper bound obtained from other solutions of the problem. Though such a bound is not available in the Healy algorithm it is used here for comparative purposes. It is assumed that enumeration of the remaining 16 lattice points would have required 10 minutes. The same type of adjustment is made for the number of LP's and iterations.

3/ Driebeek's program was halted after enumerating 100 of the 128 "unbounded" lattice points in 44 minutes. Enumeration of the remaining 28 lattice points is assumed to have required 12 minutes. The same type of adjustment is made for the number of LP's and iterations.

4/ The Davis-Kendrick-Weitzman algorithm.

5/ Option 2 was used for all the runs of the DKW algorithm reported here.

Sources of Problems

Problem Number	Source
1	Kendrick (1967)
2	Markowitz and Manne (1957)
3	Ronald E. Davis, B.A. Thesis, Committee on Applied Mathematics, Harvard University, 1967
4 thru 7	Balas, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," Operations Research, Volume 13, #4, Problems 1 through 4
8	Haldi, John, "Twenty-Five Integer Programming Test Problems," Working Paper #43, Graduate School of Business, Stanford University, December 1964, Test Problem #9. The LIP1 algorithm is reported on in Haldi, John and Leonard M. Isaacson, "A Computer Code for Integer Solutions to Linear Programs," Operations Research, Volume 13, #6, November 1965, p. 946.

BIBLIOGRAPHY

1. Balinski, M. L. (1965) "Integer Programming Methods -- A Survey," Management Science, 1965.

2. Dantzig, George B., (1963) Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey, 1963.

3. Davis, Ronald E., (1967) Program Description for MFOR - (0-1) MIP: A Code for Zero-One Mixed Integer Programming Problems, (mimeo), Memorandum No. 71, Project for Quantitative Research in Economic Development, Center for International Affairs, Harvard University, Cambridge; September, 1967.

4. Driebeek, Norman J., (1966) "An Algorithm for the Solution of Mixed Integer Programming Problems," Management Science, Vol. 12, No. 7, March, 1966.

5. Healy, W. C., Jr., (1964) "Multiple Choice Programming," Operations Research Journal, Vol. 12, No. 1, January-February 1964, pp. 122-138.

6. Kendrick, David A., (1967) Programming Investment in the Process Industries, M.I.T. Press, Cambridge, Mass., 1967.

7. Land, A. H. and Doig, A. G., (1960) "An Automatic Method of Solving Discrete Programming Problems," Econometrica, 28 No. 3, July, 1960, pp. 397-520.

8. Lawler, E. L., and Wood, D. E., (1966) "Branch and Bound Methods, a Survey," Operations Research, Vol. 14, July-August 1966.

9. Little, et al., (1963) "An Algorithm for the Traveling Salesman Problem," Operations Research Journal, Vol. 11, No. 6, November-December 1963, pp. 972-989.

10. Markowitz, H. M. and A. S. Manne, (1957) "On the Solution of Discrete Programming Problems," Econometrica, 1957.