

**A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem**



Fred Glover

*Operations Research*, Vol. 13, No. 6 (Nov. - Dec., 1965), 879-919.

Stable URL:

<http://links.jstor.org/sici?sici=0030-364X%28196511%2F12%2913%3A6%3C879%3AAMAFTZ%3E2.0.CO%3B2-J>

*Operations Research* is currently published by INFORMS.

---

Your use of the JSTOR archive indicates your acceptance of JSTOR's Terms and Conditions of Use, available at <http://www.jstor.org/about/terms.html>. JSTOR's Terms and Conditions of Use provides, in part, that unless you have obtained prior permission, you may not download an entire issue of a journal or multiple copies of articles, and you may use content in the JSTOR archive only for your personal, non-commercial use.

Please contact the publisher regarding any further use of this work. Publisher contact information may be obtained at <http://www.jstor.org/journals/informs.html>.

Each copy of any part of a JSTOR transmission must contain the same copyright notice that appears on the screen or printed page of such transmission.

---

JSTOR is an independent not-for-profit organization dedicated to creating and preserving a digital archive of scholarly journals. For more information regarding JSTOR, please contact [support@jstor.org](mailto:support@jstor.org).

# Operations Research

November-December 1965

## A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem†

Fred Glover

*Carnegie Institute of Technology, Pittsburgh, Pennsylvania*

(Received February 9, 1965)

Following a line of approach recently applied to the 0-1 integer programming problem with some success by EGON BALAS, the algorithm of this paper is based upon an underlying tree-search structure upon which a series of tests is superimposed to exclude large portions of the tree of all possible 0-1 solutions from examination. In our method, the specific design of the enumeration and tests, supplemented by the use of a special type of constraint called a 'surrogate constraint,' results in an algorithm that appears to be quite efficient in relation to other algorithms currently available for solving the 0-1 integer programming problem. Early indications of efficiency must, however, be regarded as suggestive rather than conclusive, due to the limited range and size of problems so far examined. Following the analytical development of the method, three example problems are solved in detail with the Multiphase-Dual Algorithm to illustrate various aspects of its application. An extension of the algorithm to the general integer programming problem in bounded variables is briefly sketched in a concluding section.

**T**HE PROBLEM to be solved by the algorithm of this paper may be formulated

$$\begin{array}{ll} \text{minimize} & wb, \\ \text{subject to} & wA \geq c, \quad w \geq 0, \end{array} \quad (1)$$

where  $w$  is a  $1 \times m$  row vector of 0-1 variables,  $b$  is an  $m \times 1$  column vector,  $c$  is a  $1 \times n$  row vector, and  $A$  is an  $m \times n$  matrix. The foregoing, in the

† This report was prepared as part of the activities of the *Management Sciences Research Group*, Carnegie Institute of Technology, under Contract Nonr 760(24), NR 047-048, with the U.S. Office of Naval Research.

absence of 0-1 requirements on the variables, is the customary dual linear programming formulation.

Algorithms for solving (1) when integer requirements are explicit may be divided into four groups: (i) those that use cutting-plane techniques or in a more general fashion employ integer linear transformations of the problem variables, (ii) those that employ parallel shifts of the objective function hyperplane, (iii) those based upon Boolean algebra, (iv) those that use combinatorial methods for enumerating a restricted subset of possible integer solutions.† In the first category are the well-known algorithms of R. GOMORY,<sup>[18,19,20]</sup> and the more recently developed methods of A. BEN-ISRAEL AND A. CHARNES,<sup>[2]</sup> R. YOUNG,<sup>[28]</sup> AND F. GLOVER.<sup>[11,12]</sup> Constraints designed to be used with such methods to ensure that the variables will assume integer values in the final solution have been proposed by DANTZIG,<sup>[6]</sup> ‡ CHARNES AND COOPER,<sup>[4]</sup> GOMORY,<sup>[18,20]</sup> BEN-ISRAEL AND CHARNES,<sup>[2]</sup> AND GLOVER.<sup>[13]</sup>

The second class of algorithms is associated with the names of LAND AND DOIG,<sup>[23]</sup> SZWARC,<sup>[26]</sup> ELMAGHRABY,<sup>[7]</sup> AND G. L. THOMPSON.<sup>[27,28]</sup> An important feature of these methods is that the mixed-integer linear programming problem may be treated as a direct extension of the pure integer problem.

The Boolean algebra methods of the third class are due to R. FORTET<sup>[8]</sup> AND R. CAMION.<sup>[9]</sup>

The algorithm of this paper belongs to the fourth class, which is both older and newer than the other three. Various forms of enumeration have been proposed for solving integer programming problems—particularly the 0-1 problem—since they became of interest. Only recently, however, has there appeared any method in this area sophisticated enough to make a valid claim to efficiency. Two such methods, the first devised for a special 0-1 problem that arises in the selection of prime implicants, and the second devised for the knapsack problem, have been developed respectively by GORDON, HOUSE, LECHLER, NELSON, AND RADO<sup>[22]</sup> § and by GILMORE AND GOMORY.<sup>[10]</sup> For the general 0-1 problem, a pioneering contribution was made by EGON BALAS,<sup>[1]</sup> whose additive algorithm demonstrated the capability of solving a number of problems in less time

† In point of fact, nearly all of these algorithms may be considered to employ combinatorial techniques for guaranteeing convergence to an optimal solution. While this is perhaps more conspicuous for the members of class (ii), which employ tree-search methods as an integral part of the solution strategy, the lexicographically ordered solution sequences of the algorithms of class (i) also fall into this framework.

‡ GOMORY AND HOFFMAN<sup>[21]</sup> have shown that an algorithm based solely upon the constraints of reference 6 will not in general converge.

§ The work of reference 22 as it relates to this paper is primarily due to L. D. Nelson, and is extended in reference 25.

than required to solve the corresponding linear programs, in which the variables are free to assume fractional values.

The algorithm of this paper, which may be regarded as an extension of the work of Balas, employs methods for circumscribing the underlying enumerative process that appear to be still more effective, eliminating for some problems between one-half and two-thirds of the computation required with the additive algorithm—the difference apparently growing larger as the size of the problem increases. Two ‘ill-behaved’ problems reported to have been solved by hand in  $1\frac{1}{2}$  hours and “somewhat more than four hours” with Balas’ method, yielded to the present method in 17 minutes and 85 minutes, respectively.† A handful of problems solved on the computer with other integer programming algorithms have also been solved by hand with the present method, requiring in general only a fraction of the computational effort required by the other methods. It is to be stressed, however, that the problems on which the preceding comparisons are based have been small, most of them involving between 8 and 15 variables and a like number of constraints. Thus no real conclusions can be drawn about the performance of the method for most problems of practical size. In addition, for certain types of problems that are particularly difficult to handle with other methods, it has been found that the algorithm may obtain a near-optimal solution with modest computational expenditure, but require a much larger amount of computation to obtain the optimum—or to verify that the optimum has been found. A 15 variable 36 inequality problem that was unsolved after 400 pivots with Gomory’s all integer method, and which required over 1000 pivots to solve with the stopped simplex method of G. L. Thompson, offers a notorious example. The present algorithm obtained the optimal solution in less than 30 minutes by hand, but by conservative estimate would require computation in the vicinity of that required by the stopped simplex method to verify the status of the solution.

On the other hand, a model capital budgeting problem that required several hundred pivots to solve with the stopped simplex method,‡ and which was handled much less efficiently with other methods of the same class, has been solved by hand with the present algorithm in an hour and a half.

Evaluation of such results must be tempered in view of the limited

† This statement must be qualified by the fact that the computation times that are being compared were achieved by different persons. Moreover, our remarks are not intended to imply the absence of problems for which the additive algorithm does better than the method of this paper. Several such problems have, in fact, been found.

‡ I am indebted to PROF. G. L. THOMPSON for making these and other computational results obtained with his stopped simplex method available to me.

range, as well as the size, of problems examined. Moreover, the fact that our algorithm appears to obtain quite good results in relation to methods designed to handle the general integer programming problem does not offer a basis for comparing the merits of our algorithm to other methods when the variables may assume integer values greater than one. We shall, however, sketch an extension of the Multiphase-Dual Algorithm to the general case in a later section, and provide explicit comparisons for the more general integer programming problem in a subsequent paper.

### THE ENUMERATIVE SUBSTRUCTURE

THE UNDERLYING enumeration procedure upon which the algorithm of this paper is superimposed is that of elementary tree search. Some path along the tree of solutions is traced until either a new solution is obtained or a node is reached which yields information that all solutions in which that particular node is included may be ruled out of consideration. Thereupon the process backtracks to the unique node that immediately precedes the one ruled out, and embarks on a different path, unless none are left and it becomes necessary to backtrack further. Once the process is pushed back to the starting node, and information is obtained that forbids tracing out any more branches of the tree, the procedure terminates.

This is the usual method employed in enumerating sets of logical possibilities and has the principal advantage that it requires very little memory to execute. It is obviously not the only method (other procedures will be discussed later), but its directness commends it for initial consideration, and it is the basis upon which the example problems in this paper have been solved.

As a class of problem-solving techniques, tree-search methods have been employed for a long time, both as heuristics and algorithms. On the heuristic level, a form of restricted tree search has been applied to diverse problems ranging all the way from chess to production scheduling. On the algorithmic level tree search has been applied to the traveling salesman problem<sup>[24]</sup> and the job shop scheduling problem,<sup>[9]</sup> as well as to the integer programming problem.

Specifically, we call the tree-search algorithm of this paper a multiphase-dual method because the solution process is decomposed into a series of phases, each one applied to a problem derived from problem (1) (in dual formulation) under the assumption that certain of the problem variables are assigned specific values. The phases are essentially identical to one another except for the problems to which they are applied. During any particular phase, the current problem is either solved or a new problem is created to be handled by the next phase.

To illustrate the underlying structure of the algorithm more precisely,

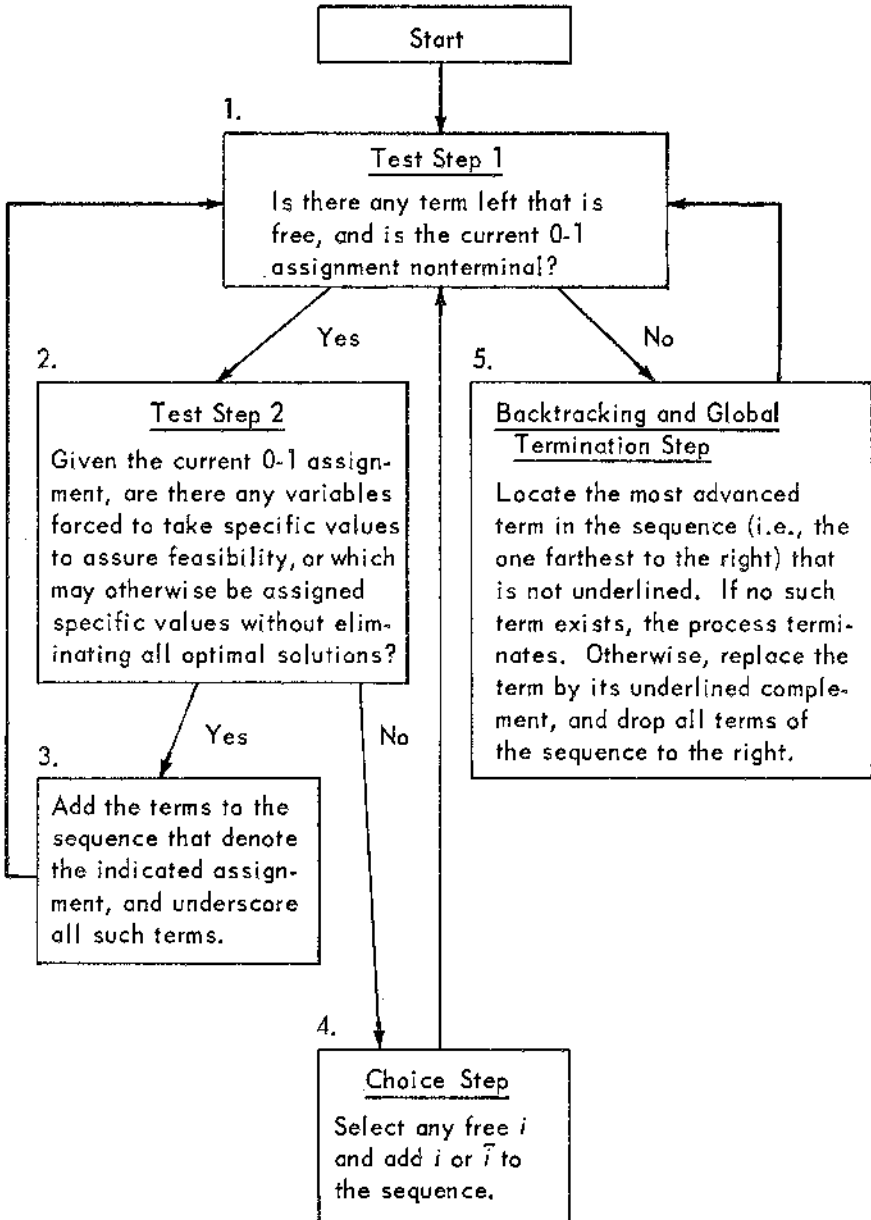
we will use the following notation and conventions. The term  $i$  will be used to denote  $w_i=1$ , and the complementary term  $\bar{i}$  will be used to denote  $w_i=0$ . We define a *solution sequence* to be a sequence of such terms in which (i) no term appears more than once, (ii) the corresponding 0-1 assignment to some or all of the problem variables is well-defined (i.e., not both  $i$  and  $\bar{i}$  can appear in the sequence, and for each  $i$  such that  $i$  or  $\bar{i}$  is in the sequence,  $i$  is an integer satisfying  $1 \leq i \leq m$ ). For example, if  $m \geq 5$ , then  $3, 4, \bar{2}, 5$  is a solution sequence associated with the assignment  $w_3=1, w_4=1, w_2=0$ , and  $w_5=1$ .

The term  $i$ —or alternately the variable  $w_i$ —will be said to be *free* if neither  $i$  nor  $\bar{i}$  appears in the solution sequence. Thus, in the foregoing sequence the free terms are  $i=1$  and  $i=6, 7, \dots, m$ .

An underlined subsequence of terms will indicate that the partial 0-1 assignment associated with the underlined terms is implied by the solution sequence to the left of the underlined terms. For example,  $3, 4, \underline{\bar{2}}, \underline{5}$  gives the same solution sequence as above, except here the underlining indicates that  $w_2=0$  and  $w_5=1$  are implied by  $w_3=1$  and  $w_4=1$ . Such a situation would arise if the problem contained the constraint  $1w_1 - 7w_2 - 2w_3 - 2w_4 + 5w_5 \geq 1$ , since  $w_2$  must equal 0 and  $w_5$  must equal 1 in order to satisfy the constraint when  $w_3=w_4=1$ . In fact,  $w_2=0$  is implied by this constraint regardless of the values assumed by the other variables, and  $w_5=1$  is implied when either  $w_3=1$  or  $w_4=1$ . Thus, a second solution sequence compatible with this constraint is  $\underline{\bar{2}}, 3, \underline{5}, 4$ . Note also that if the constant term on the right-hand side of the inequality sign in the foregoing constraint were 4 instead of 1, the solution sequence  $\underline{\bar{2}}, 3$  would not be compatible with any feasible 0-1 assignment. Under such conditions the sequence  $\underline{\bar{2}}, 3$ —and the associated 0-1 assignment—will be called (locally) *terminal*. In general, we define a *terminal solution sequence* to be one for which there exists no 0-1 assignment of the free variables that will produce a feasible solution to problem (1) or to an augmented problem derived from (1) by adjoining certain additional constraints in the course of solving for the optimum. We require of the augmented problem, however, that the additional constraints must not render all optimal solutions to (1) infeasible unless, in fact, at least one optimal solution has already been obtained.† Thus, for example a special case of a terminal solution sequence

† An exception occurs when certain restrictive constraints are adjoined to the problem at various stages as a means of partitioning the solution space into disjoint subsets. With each such constraint is associated a 0-1 variable that takes the value 1 when the constraint is required to hold and the value 0 when the constraint is required not to hold. The particular application of this procedure in the present paper is confined to those constraints that require one of the original problem variables to equal 0 or 1, as reflected directly by the terms of the solution sequence. The details of the development and application of the procedure in its more general form, however, are reserved to reference 16.

Diagram 1



is one that is incompatible with any solution that satisfies the constraint  $wb \leq b_0$ , where  $b_0$  is a known upper bound on the optimal objective function value for problem (1). It is apparent that whenever a feasible solution to (1) is obtained, a value for  $b_0$  is thereby given so that the constraint  $wb \leq b_0$  may be added to the other constraints to assist in the search for an improved solution.

The mechanism used to determine whether a given solution sequence is locally terminal, or whether some of the problem variables are compelled to assume specific values, will be called a *test*. Simple versions of three of the tests to be developed later may be inferred directly from the preceding discussion, as we will elaborate upon and clarify shortly.

The left-right orientation of the solution sequence is the key that provides the basis for the underlying structure of the algorithm. We now specify this structure in Diagram 1.

**THEOREM 1.** *For any problem (1) in 0-1 variables, the process specified in Diagram 1 will generate a nonrepeating (and hence finite) sequence of 0-1 solutions. If the feasible solution set of (1) is nonempty, then at least one of the solutions generated will be optimal.*

The preceding theorem, and all others in this paper, will be proved in the appendix.

The nature of the foregoing method may be illustrated by examining the interpretation to be accorded to a specific solution sequence. Suppose, for example, that the solution sequence  $3, \bar{2}, 4, 5$  is generated at some phase in the application of the foregoing method. Then it follows that: (i) all nonterminal sequences that begin  $3, 2$  and  $3, \bar{2}, 4, \bar{5}$  have already been examined or otherwise ruled out of consideration, (ii) all nonterminal and otherwise nonexcluded solutions in which  $w_3 = 1$  will be examined before examining any in which  $w_3 = 0$ , and (iii) all nonterminal and nonexcluded solutions in which  $w_3 = 1, w_2 = 0$ , and  $w_4 = 1$  will be examined before examining any in which  $w_3 = 1, w_2 = 0$ , and  $w_4 = 0$ .

In the absence of any tests to identify terminal solutions, or to determine whether certain variables may be forced to equal 0 or 1, the solution process of course reduces to complete enumeration. To give a clearer idea of how this process might work under a rigid rule of choice, we will illustrate the enumerative base when  $m = 3$ . For definiteness, assume that the least  $i$  which is free will always be selected to be added to the sequence of the choice step. The eight 0-1 assignments to the variables  $w_1, w_2$ , and  $w_3$  that are generated by the process are then as follows:

$$\begin{array}{ll} 1, 2, 3 & \bar{1}, 2, 3 \\ 1, 2, \bar{3} & \bar{\bar{1}}, 2, \bar{3} \\ 1, \bar{2}, 3 & \bar{1}, \bar{2}, 3 \\ 1, \bar{2}, \bar{3} & \bar{\bar{1}}, \bar{\bar{2}}, \bar{3} \end{array}$$

Without more advanced methods to raise the process of Diagram 1 above the level of enumeration, the algorithm would be not only trivial but grossly inefficient. Fortunately, methods of considerable power, reapplied at each phase of the solution process, are available to provide the efficiencies desired. We have already alluded to the tests, which will be specified in detail in the section "Tests Used with the Algorithm." Basic to several of these tests, and designed to give them more effective application, is the surrogate constraint, to which we now turn our attention.

### THE SURROGATE CONSTRAINT†

ONE OF THE central features of the algorithm of this paper is the use of a constraint that enforces restrictions upon the optimal solution to (1) which can not be determined from any of the individual constraints of (1) in isolation. This constraint is called a surrogate constraint (*s*-constraint) since its function is to serve as a substitute for some of the original problem constraints in guiding the progress of the algorithm.

The *s*-constraint is defined as a nonnegative linear combination of the constraints of (1) in which at least one of the constraints of (1) is given a positive weight,‡ and will be represented by

$$wa \geq c_0,$$

where  $a = (a_1 a_2 \cdots a_m)^T$  is an  $m \times 1$  column vector and  $c_0$  is a scalar. Thus, in matrix notation,  $a = Au$  and  $c_0 = cu$ , where  $u = (u_1 u_2 \cdots u_n)^T$  is an  $n \times 1$  column vector such that  $u \geq 0$  and  $u_i > 0$  for at least one  $i$ ,  $1 \leq i \leq n$ . It may be noted that each of the constraints of (1) is a special case of the *s*-constraint, obtained by assigning a weight of 1 to the indicated constraint and by assigning all other constraints a weight of 0.

To obtain an *s*-constraint, or a set of *s*-constraints, which may be useful for solving (1), consider the problem§

$$\begin{array}{ll} \text{minimize} & wb, \\ \text{subject to} & wa \geq c_0, \quad w_i = 0, 1. \end{array} \quad (2)$$

By the definition of the *s*-constraint, we may immediately state the following

† I would like to thank PROF. G. L. THOMPSON for comments and suggestions which I have used to improve the presentation of this section.

‡ A logical extension of this definition would be to include any constraints in the linear combination that may be implied by  $wA \geq c$  in order to assure that the solution to (1) will be in 0-1 integers.

§ There is a close relation between this problem and the bounded variable knapsack problem. See references 5 and 14.

LEMMA.† (i) If  $\bar{w}$  is a feasible 0-1 solution to (1), then  $\bar{w}$  is also feasible for (2).

(ii) If (2) has no feasible solution, then neither does (1).

(iii) If  $\bar{w}$  is feasible for (2), then it must satisfy at least one of the constraints assigned a nonzero weight in defining  $w \geq c_0$ . If  $\bar{w}$  is feasible for (2) and infeasible for (1), and if at least one of the unsatisfied constraints of (1) is assigned a positive weight in defining the  $s$ -constraint, then there exists a constraint  $wA_j \geq c_j$  of problem (1) such that  $\bar{w}A_j > c_j$ , where  $A_j$  denotes the  $j$ th column of the  $A$  matrix.

(iv) If  $\bar{w}$  is optimal for (2) and  $w^*$  is optimal for (1), then  $\bar{w}b \leq w^*b$ .

(v) If  $\bar{w}$  is optimal for (2) and feasible for (1), then  $\bar{w}$  is optimal for (1).

Conclusion (iv) of the lemma may be used to develop a criterion for a 'strong'  $s$ -constraint. Given two  $s$ -constraints derived from the same problem, we will say that the first is *stronger* if the optimal solution to (2) obtained with that constraint yields a greater value for  $wb$  than the optimal solution to (2) obtained with the second. To derive a useful principle that will assist in determining a strong  $s$ -constraint in the sense in which we use the word 'strong' here, we examine the situation in which (1) consists of only two constraints.

THEOREM 2. Let  $A$  consist of the two column vectors  $A_1$  and  $A_2$ , let  $c = (c_1 \ c_2)$ , and let  $u = (u_1 \ u_2)^T$ . Suppose that when  $u = \bar{u} > 0$  the vector  $\bar{w}$  is a feasible solution to (2) ( $\bar{w}A\bar{u} \geq c\bar{u}$ ), but  $\bar{w}A_2 < c_2$ . Then, maintaining  $u > 0$ ,  $\bar{w}$  is a feasible solution to (2) for  $0 < u_2/u_1 \leq P$ , and infeasible for (2) when  $u_2/u_1 > P$ , where  $P = (\bar{w}A_1 - c_1)/(c_2 - \bar{w}A_2)$ .

To interpret Theorem 2, note that by (iii) of the lemma above, the assumptions of the theorem imply  $\bar{w}A_1 > c_1$ . Thus the quantity  $\bar{w}A_1 - c_1$  may be described as the (positive) amount by which  $\bar{w}$  oversatisfies the constraint  $wA_1 \geq c_1$ . Similarly,  $c_2 - \bar{w}A_2$  is the (positive) amount by which  $\bar{w}$  undersatisfies  $wA_2 \geq c_2$ . The theorem says then that the solution  $\bar{w}$  will remain feasible for (2) as long as the weight assigned to the unsatisfied constraint of (1), divided by the weight assigned to the satisfied constraint of (1), stays less than or equal to  $P$ , but will no longer be feasible for (2) if this ratio becomes larger than  $P$ , where  $P$  is the ratio of the amount by which  $\bar{w}$  oversatisfies  $wA_1 \geq c_1$  to the amount by which it undersatisfies  $wA_2 \geq c_2$ .

To make use of this result in determining a strong  $s$ -constraint for (1), we proceed as follows. Let  $\bar{w}$  denote a feasible solution to problem (2), where (2) is defined relative to a given  $s$ -constraint  $S$ . Then  $S$  may be divided into the two component constraints  $F$  and  $G$ , where  $F$  is a linear

† The proof of the lemma is included in the appendix along with the proofs of the theorems.

combination of those constraints of (1) that are satisfied by  $\bar{w}$ , and  $G$  is a linear combination of those constraints of (1) that are unsatisfied by  $\bar{w}$ —the weight given to each constraint composing  $F$  and  $G$  being the same weight as the constraint had in the  $s$ -constraint  $S$ . (Hence  $S = F + G$ .) Further, let  $f$  denote the amount by which  $\bar{w}$  oversatisfies  $F$ , and  $g$  denote the amount by which  $\bar{w}$  undersatisfies  $G$ . (Since  $S$  is satisfied by  $\bar{w}$ , we may note that  $f \geq g$ .) By the preceding theorem the  $s$ -constraint  $F + kG$  will be satisfied by  $\bar{w}$  for  $0 < k \leq f/g$ , and will fail to be satisfied by  $\bar{w}$  when  $k = f/g + \epsilon$ , where  $\epsilon$  is a small positive number (by the correspondence  $f/g = P$ ,  $u_1 = 1$ , and  $u_2 = k$ ).<sup>†</sup> Using these results, we now outline a method that gives the basis for the method of  $s$ -constraint determination to be used in this paper.

**THEOREM 3.** *Assume that  $A$  and  $c$  are given as in Theorem 2, and that problem (1) has a feasible solution. Assume, moreover, that the cutoff rule for Method A specifies termination when the constraint of problem (1) that was unsatisfied by the first  $\bar{w}$  becomes satisfied by the current  $\bar{w}$ . Then there exists a positive value for  $\epsilon$  for which Method A will produce an  $s$ -constraint that is as strong as any  $s$ -constraint that can be obtained from (1).*

Since a permissible value for  $\epsilon$  is not known in advance, it is to be noted that it is possible to locate the desired  $s$ -constraint [when problem (1) consists of two constraints] by starting with  $\epsilon$  at any positive value and then decreasing  $\epsilon$  when Method A would otherwise come to a halt, until no such decrease is possible while maintaining  $\epsilon > 0$ .

When problem (1) consists of more than two constraints the results of Theorem 3 do not apply, and for computational expediency the method of  $s$ -constraint determination that we will use in this paper (Method B) differs from the version of Method A used in Theorem 3 in two respects: (i) the cutoff rule will simply be to halt after a given number of  $s$ -constraints have been generated, (ii) an approximation to the optimal 0-1 solution to (2) will be used in place of the optimal solution to (2).

To illustrate Method B, we must first derive the indicated approximation to the optimal solution of (2). In the process, theorems will be stated that will provide an important foundation for subsequent portions of the paper. The first of these theorems, while very nearly evident, allows an important simplification upon which subsequent results may be based.

**THEOREM 4.** *If (2) has a feasible fractional solution,<sup>†</sup> then there exists an*

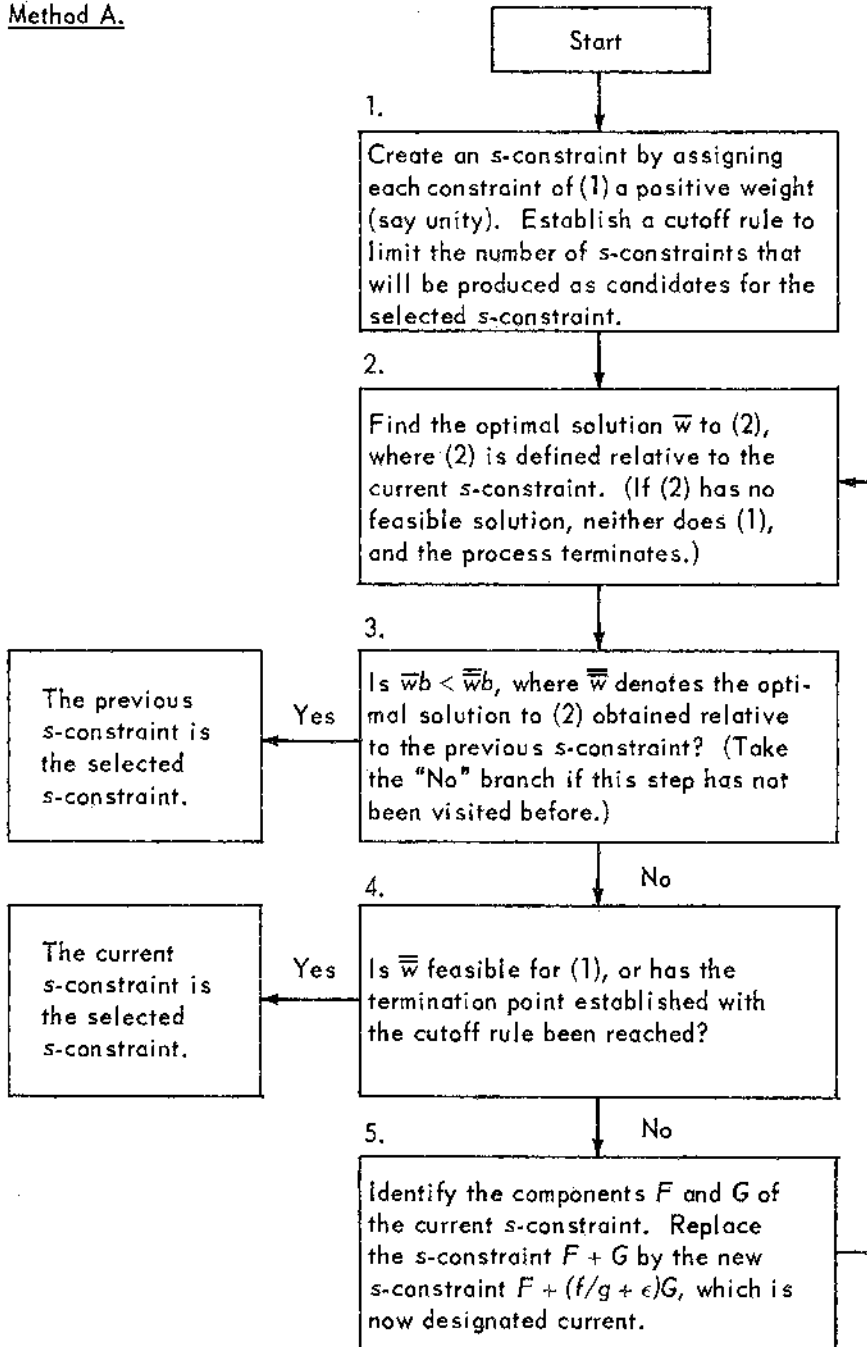
<sup>†</sup> More precisely, if  $S$  denotes the constraint  $wA\bar{u} \geq c\bar{u}$ , then  $F$  denotes the constraint  $wA\bar{u}^1 \geq c\bar{u}^1$  and  $G$  denotes  $wA\bar{u}^2 \geq c\bar{u}^2$ , where  $\bar{u}^1$  and  $\bar{u}^2$  are defined so that

$$\bar{u}_j^1 = \begin{cases} \bar{u}_j & \text{if } \bar{w}A_j \geq c_j \\ 0 & \text{otherwise} \end{cases}, \text{ and } \bar{u}^2 = \bar{u} - \bar{u}^1.$$

Then  $f = \bar{w}A\bar{u}^1 - c\bar{u}^1$ ,  $g = c\bar{u}^2 - \bar{w}A\bar{u}^2$ , and  $F + kG$  denotes the constraint  $wA\bar{u} \geq c\bar{u}$ , where  $\bar{u} = \bar{u}^1 + k\bar{u}^2$ .

<sup>†</sup>By a fractional solution we mean one in which  $0 \leq w_i \leq 1$  for  $i = 1, \dots, m$ , but  $w$  is not constrained to be integer.

## Method A.



optimal fractional and an optimal integer solution to (2) in which the following assignment of values occurs:

$$w_i = \begin{cases} 1 & \text{if } b_i \leq 0 \text{ and } a_i \geq 0, \\ 0 & \text{if } b_i \geq 0 \text{ and } a_i \leq 0. \end{cases}$$

(When  $b_i = a_i = 0$ , the value of  $w_i$  is immaterial.)

Using Theorem 4, problem (2) thus reduces to a simpler problem in which, for each  $i$ , either  $b_i a_i > 0$  or  $b_i a_i < 0$ . But in fact we may assume that  $b_i a_i > 0$  for all  $i$ , or alternately that  $b_i a_i < 0$  for all  $i$ , by noting that a problem in 0-1 variables equivalent to (2) is obtained by substituting  $v_i = 1 - w_i$  for any subset of the  $w_i$ , and setting  $v_i = w_i$  for the remainder of the  $w_i$ , yielding the problem

$$\begin{aligned} \text{minimize} & \quad v\bar{b} + \sum' b_i, \\ \text{subject to} & \quad v\bar{a} \geq \bar{c}_0, \quad v_i = 0, 1, \end{aligned}$$

where

$$\bar{a}_i, \bar{b}_i = \begin{cases} a_i, b_i & \text{if } v_i = w_i, \\ -a_i, -b_i & \text{if } v_i = 1 - w_i, \end{cases}$$

$$\bar{c}_0 = c_0 - \sum' a_i,$$

and  $\sum'$  is the summation over those  $i$  for which  $v_i = 1 - w_i$ . Here the constant term  $\sum' b_i$  is of course irrelevant to determining the solution vector which optimizes (2a).

Thus, under the assumption that Theorem 4 has been applied beforehand, and using (2a), we assure that  $a_i b_i > 0$  for all  $i$  by substituting

$$v_i = \begin{cases} w_i & \text{if } b_i a_i > 0, \\ 1 - w_i & \text{if } b_i a_i < 0. \end{cases}$$

The condition  $a_i b_i < 0$  for all  $i$  is assured similarly. To take advantage of these two alternative structures we now state the following 'dual' theorems. †  
**THEOREM 5A.** Assume  $b > 0$  and  $a > 0$ , and define an auxiliary (subscripted) indexing so that  $b_{i_p}/a_{i_p} < b_{i_q}/a_{i_q}$  implies  $p < q$ . Further, let  $r$  be the least integer ( $0 \leq r \leq m$ ) for which  $\sum_{p \leq r} a_{i_p} \geq c_0$ . ‡ Then an optimal fractional solution to (2) is given by

$$w_{i_p} = \begin{cases} 1 & \text{if } p < r, \\ 0 & \text{if } p > r, \end{cases}$$

† Theorem 5B is due to G. B. DANTZIG,<sup>[6]</sup> and follows from a more general theorem by the author in reference 14. The two theorems 5A and 5B may be put in direct correspondence by means of (2a).

‡ We will use the usual convention throughout this paper that any sum that is otherwise undefined will be equal to zero. Thus, for example,  $\sum_{p \leq r} a_{i_p} = \sum_{p > s} a_{i_p} = 0$  if  $r \leq 0$  or  $s \geq m$ .

and  $w_{i_r} = (c_0 - \sum_{p < r} a_{i_p}) / a_{i_r}$  (provided  $r > 0$ ).

If  $r$  does not exist (i.e.,  $\sum_{k \leq m} a_k < c_0$ ), then (2) has no feasible solution.

**THEOREM 5B.** Assume  $b < 0$  and  $a < 0$ , and define the auxiliary indexing as in Theorem 5A. Let  $s$  be the least integer ( $0 \leq s \leq m$ ) for which  $\sum_{p > s} a_{i_p} \geq c_0$ . Then an optimal fractional solution to (2) is given by

$$w_{i_p} = \begin{cases} 1 & \text{if } p > s, \\ 0 & \text{if } p < s, \end{cases}$$

and  $w_{i_s} = (c_0 - \sum_{p > s} a_{i_p}) / a_{i_s}$  (provided  $s > 0$ ).

If  $s$  does not exist ( $c_0 > 0$ ), then (2) has no feasible solution.

We will now illustrate the use of the preceding theorems in conjunction with (2a) to give the fractional optimal solution to problem (2).

### Example

Minimize  $3w_1 - 1w_2 - 4w_3 + 6w_4 - 5w_5 + 4w_6 - 2w_7$ ,

subject to  $4w_1 - 3w_2 - 1w_3 + 2w_4 - 3w_5 - 4w_6 + 5w_7 \geq 6$ .

Applying Theorem 4, we first set  $w_6 = 0$  and  $w_7 = 1$ , so that the problem reduces to

minimize  $3w_1 - 1w_2 - 4w_3 + 6w_4 - 5w_5 - 2$ ,

subject to  $4w_1 - 3w_2 - 1w_3 + 2w_4 - 3w_5 \geq 1$ .

If Theorem 5A is elected to solve this problem, we use the substitutions

$$v_i = \begin{cases} w_i & \text{if } i = 1, 4, \\ 1 - w_i & \text{if } i = 2, 3, 5, \end{cases}$$

yielding

minimize  $3v_1 + 1v_2 + 4v_3 + 6v_4 + 5v_5 - 12$ ,

subject to  $4v_1 + 3v_2 + 1v_3 + 2v_4 + 3v_5 \geq 8$ ,

auxiliary index  $p$ :  $\begin{matrix} 2 & 1 & 5 & 4 & 3. \end{matrix}$

The auxiliary index  $p$  is listed beneath the variable to which it corresponds, in accordance with the definition given in Theorem 5A. We see from this that  $r = 3$ , and that the solution determined from Theorem 5A is  $v_1 = v_2 = 1$ ,  $v_5 = 1/3$ , and  $v_3 = v_4 = 0$ . Translated back in terms of the original variables this yields  $w_1 = w_3 = 1$ ,  $w_5 = 2/3$ , and  $w_2 = w_4 = 0$ . Alternately, applying Theorem 5B, we use the substitutions

$$v_i = \begin{cases} w_i & \text{if } i = 2, 3, 5, \\ 1 - w_i & \text{if } i = 1, 4, \end{cases}$$

which yields

$$\begin{array}{ll} \text{minimize} & -3v_1 - 1v_2 - 4v_3 - 6v_4 - 5v_5 + 7, \\ \text{subject to} & -4v_1 - 3v_2 - 1v_3 - 2v_4 - 3v_5 \geq -5, \\ \text{auxiliary index } p: & \quad 2 \quad 1 \quad 5 \quad 4 \quad 3. \end{array}$$

It may be observed that the auxiliary indexing here corresponds exactly to that of the preceding case, and also that  $s=r=3$ . It can readily be shown that this will always happen if ties in the auxiliary indexing are broken properly. (At this point, we allow such ties to be broken in any fashion.) The optimal fractional solution for the above problem prescribed by Theorem 5B is  $v_3=v_4=1$ ,  $v_5=2/3$ , and  $v_1=v_2=0$ , which of course yields the same solution in terms of the original problem as obtained by Theorem 5A.

The foregoing results will now be used to obtain an approximation to the optimal integer solution to (2), which may be used with Method B to determine an  $s$ -constraint for problem (1). It evidently suffices to determine such an approximation for the conditions specified in Theorems 5A and 5B.†

*Approximation 1.* (For  $b>0$  and  $a>0$ ). Begin with  $q=r-1$ , and then decrease  $q$  by integer steps to 1, removing each  $a_{i_q}$  from  $\sum_{p \leq r} a_{i_p}$ , which allows the sum of the remaining terms—excluding those already removed—to equal or exceed  $c_0$ . Then set

$$w_{i_p} = \begin{cases} 0 & \text{if } p > r \text{ or if } a_{i_p} \text{ was removed from the summation,} \\ 1 & \text{otherwise.} \end{cases}$$

*Approximation 2.* (For  $b<0$  and  $a<0$ ) If  $s=0$ , let  $w_i=1$  for all  $i$ . Otherwise, define  $q$  so that  $b_{i_q} = \max\{b_{i_p} | p \geq s \text{ and } a_{i_p} \leq \sum_{p \geq s} a_{i_p} - c_0\}$ . ( $q$  is well-defined since  $p=s$  satisfies the restrictions.) Then let

$$w_{i_p} = \begin{cases} 0 & \text{if } p = q \text{ or if } p < s, \\ 1 & \text{otherwise.} \end{cases}$$

While Approximations 1 and 2 are not equivalent, it may be verified from the definitions of  $r$  and  $s$  that they each yield a feasible integer solution or (2) if one exists. Before using the above approximations to illustrate Method B for  $s$ -constraint determination, we characterize a special case of some interest in which Approximations 1 and 2 both yield an optimal solution to (2).‡

**THEOREM 6A.** Assume  $a, b>0$  and that  $a_{i_p} > \sum_{k \leq r} a_{i_k} - c_0$  for all  $p < r$ .

† Although it is always possible by means of (2a) and Theorem 4 to deal only with  $a, b>0$  or with  $a, b<0$ , it is computationally useful to have results for both cases available directly.

‡ To assure the validity of this statement when there are ties to be resolved in the auxiliary indexing, and to give Theorems 6A and 6B the broadest application, we impose the additional restriction on the auxiliary indexing that  $b_{i_p}/a_{i_p} = b_{i_q}/a_{i_q}$  in conjunction with  $|\alpha_{i_p}| > |\alpha_{i_q}|$  implies  $p > q$ .

Then if  $a_{i_q} \leq a_{i_p}$  and  $b_{i_q} \geq b_{i_p}$  for each  $p \leq r$  and for each  $q > r$ , the optimal 0-1 solution to (2) is given by

$$w_{i_p} = \begin{cases} 1 & \text{if } p \leq r, \\ 0 & \text{if } p > r. \end{cases}$$

THEOREM 6B.† Assume  $a, b < 0$  and that  $a_{i_q} < c_0 - \sum_{k>s} a_{i_k}$  for all  $q < s$ . Then if  $a_{i_q} \leq a_{i_p}$  and  $b_{i_q} \geq b_{i_p}$  for each  $q \leq s$  and for each  $p > s$ , the optimal 0-1 solution to (2) is given by

$$w_{i_p} = \begin{cases} 1 & \text{if } p > s, \\ 0 & \text{if } p \leq s. \end{cases}$$

Several important problems that occur in integer programming fall within the jurisdiction of the foregoing theorems. When  $b_i = 1$  for all  $i$ , or when  $b_i = -1$  for all  $i$ , the conditions of Theorems 6A and 6B will be met regardless of the  $s$ -constraint employed (provided Theorem 4 is used to eliminate all variables in which  $b_i$  and  $a_i$  do not have the same sign). By the use of problem (2a), any problem in which  $b_i = \pm k$  falls in the same category.

More extensive application of the preceding theorems will be made in the section to follow. We now turn to Method B for  $s$ -constraint determination outlined earlier.

The following example illustrates the use of Method B in conjunction with Approximation 1.

*Example*

Minimize  $1w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6 + 7w_7,$   
 subject to  $2w_1 + 5w_2 + 3w_3 + 2w_4 + 3w_6 + 4w_6 + 7w_7 \geq 8,$   
 $-1w_1 + 0w_2 + 3w_3 + 2w_4 - 1w_5 + 3w_6 + 5w_7 \geq 7,$   
 $1w_1 + 0w_2 - 2w_3 + 3w_4 + 1w_5 - 1w_6 + 5w_7 \geq 5.$

For convenience in this and subsequent illustrations, problem (1) will be summarized in the following tableau form.‡

$-b$	$A$
	$c$

† Using problem (2a), Theorems 6A and 6B assert the same thing.  
 ‡ This form is used, with the  $-b$  vector in the first column, so that the constraint  $wb \leq b_0$  may later be summarized by this column in the same way that the remaining constraints are represented in the tableau—i. e.,  $w(-b) \geq -b_0$ .

Thus, the example problem is given by the tableau on the left below.

				Auxiliary index $p$					
				$S_1$	$S_2$	$S_3$	$S_1$	$S_2$	$S_3$
-1	2	-1	1	2	0.9	2.05	3	6	4
-2	5	0	0	5*	5.	5.	1	2	3
-3	3	3	-2	4	7.3*	5.	5	3	5
-4	2	2	3	7	9.2	12.65*	4	4	2
-5	3	-1	1	3	1.9	3.05	7	7	7
-6	4	3	-1	6	9.3	8.15	6	5	6
-7	7	5	5	17*	22.5*	28.25*	2	1	1
	8	7	5	20	27.7	33.45			

For this example we have used a value of 0.1 for  $\epsilon$ , and used the cutoff rule that specifies termination after the fifth  $s$ -constraint generated by Method B. The stars beside the coefficients of the  $s$ -constraints  $S_1$ ,  $S_2$ , and  $S_3$  denote the variables set equal to unity in the solution obtained with Approximation 1, hence the solution vectors obtained in these three cases are respectively  $w = (0100001)$ ,  $(0010001)$ , and  $(0001001)$ .

The table on the right above lists the subscripts  $p$  determined by the auxiliary indexing rules for each of the three  $s$ -constraints. For this problem it was of course unnecessary to determine  $p$  for  $p > 2$  in  $S_1$  and  $S_2$ , or to determine  $p$  for  $p > 3$  in  $S_2$ . Tables IA, IB, and IC below summarize the data and computations used to derive the  $s$ -constraints with Method B.

For constraint  $S_1$ , we have arbitrarily set the weight of each original constraint equal to unity, as shown in the first column of Table IA. The solution to the problem based on  $S_1$  oversatisfies constraint  $C_1$  (associated with the component  $c_1$  of the  $c$  vector) by 4, undersatisfies  $C_2$  by 2, and exactly satisfies  $C_3$ , as shown in column 1 of Table IB. The second  $s$ -constraint is then obtained from this information using Table IC, in which each column represents the termwise product of the corresponding columns of Tables IA and IB. To determine  $f$ , we sum the positive components of the first column of Table IC, and for  $g$ , we sum the absolute values of the negative components. This yields  $f/g + \epsilon = 4/2 + 0.1 = 2.1$ , which we multiply times each weight of the first column of Table IA for which the corresponding constraint was unsatisfied by the first solution. These products are then entered in the second column of Table IA, as shown by the entry 2.1 for  $u_2$ . The remaining columns of the tables are generated similarly, and we select  $S_3$  to be the distinguished  $s$ -constraint to be employed in solving (1), since the third solution is feasible for the original problem. The cutoff point of 5  $s$ -constraints was superfluous in this case.

It may be noted that for computational expediency it is unnecessary to compute each  $s$ -constraint in terms of all the original constraints, but only in terms of the unsatisfied original constraints and the previous  $s$ -constraint. Thus  $S_2 = S_1 + 1.1C_2$ , and  $S_3 = S_2 + 1.15C_3$ .

While we have spoken up to now in terms of deriving a single  $s$ -constraint to assist in solving (1), such a restriction is altogether unnecessary and—in many cases—undesirable. Other  $s$ -constraints to be used in solving (1) may be obtained by the foregoing method under the explicit assumption

TABLE IA  
WEIGHTS FOR ORIGINAL CONSTRAINTS WHICH PRODUCE  $s$ -CONSTRAINTS

	$S_1$	$S_2$	$S_3$
$u_1$	1	1	1
$u_2$	1	2.1	2.1
$u_3$	1	1	2.15

TABLE IB  
AMOUNT BY WHICH CONSTRAINTS OVERSATISFIED OR UNDERSATISFIED (-).

	Solution for $S_1$	Solution for $S_2$	Solution for $S_3$
$C_1$	4	2	1
$C_2$	-2	1	0
$C_3$	0	-2	3

TABLE IC  
COMPUTATION TABLE

Product Columns		
4	2	1
-2	2.1	0
0	-2	6.45

that certain of the problem variables have been assigned definite values, or a handful of the  $s$ -constraints generated by Method B in the absence of such assumptions may also be used. We will return to this issue later, since in many cases the determination of a good set of  $s$ -constraints has an unusually powerful effect on the rate of convergence to the optimum.

#### TESTS USED WITH THE ALGORITHM

AS INDICATED in the first section, the algorithm of this paper rules branches of the solution tree from consideration by determining either (a) that no

feasible or optimal solution exists along the present branch, or (b) that some of the problem variables not already assigned a specific value may—or must—be set equal to a given value if an optimal solution does lie along the current branch.

It is also sometimes useful to employ a test to determine whether a problem constraint is locally nonbinding—that is, whether the constraint may be discarded while investigating the current branch of the solution tree without eliminating all optimal solutions along that branch. Not all nonbinding constraints will be considered for removal, since, for example, the *s*-constraints are initially in this category.

To clarify the manner in which the tests of the algorithm will be employed, note that at any stage of the solution process we may rewrite problem (1) as

$$\begin{aligned} &\text{minimize} && w^\alpha b^\alpha + w^\beta b^\beta, \\ &\text{subject to} && w^\alpha A^\alpha + w^\beta A^\beta \geq c, \quad w_i^\alpha = 0, 1 \quad \text{and} \quad w_i^\beta = 0, 1, \end{aligned}$$

where we have split the *w* vector into the two subvectors  $w^\alpha$  and  $w^\beta$  corresponding respectively to those variables that are free and those that are assigned a specific value at the current stage of the algorithm. The vector *b* and the matrix *A* have correspondingly been divided into the subvectors  $b^\alpha$  and  $b^\beta$ , and submatrices  $A^\alpha$  and  $A^\beta$ .

Suppose that  $\bar{w}^\beta$  denotes the 0-1 assignment to the variables currently given a specific value. Then the problem reduces to

$$\begin{aligned} &\text{minimize} && w^\alpha b^\alpha + \bar{w}^\beta b^\beta && (3) \\ &\text{subject to} && w^\alpha A^\alpha \geq c', \quad w_i^\alpha = 0, 1, \end{aligned}$$

where  $c' = c - \bar{w}^\beta b^\beta$ . Since the term  $\bar{w}^\beta b^\beta$  is a constant, and will have no influence on the  $w^\alpha$  vector which optimizes (3), problem (3) has essentially the same form as problem (1). We will refer to an optimal solution to (3) as a *locally optimal* solution to (1), and refer to  $c'$  as the *current c vector*. It is to be noted that whenever  $c' \leq 0$ , the *trial solution* obtained by setting  $w^\alpha = 0$  is feasible both for (3) and—by extension—for (1). Thus when  $w^\alpha = 0$  yields a feasible solution,  $\bar{w}^\beta b^\beta$  determines an upper bound for the optimal objective function value for (1). Taking note of this fact we introduce a constraint of the form  $wb \leq b_0$  [i.e.,  $w(-b) \geq -b_0$ ], which is added to the constraint set of (1) just as the *s*-constraints are added. When the objective is to find all optimal solutions to (1),  $b_0$  is set equal to the best  $\bar{w}^\beta b^\beta$  previously obtained for which  $w^\alpha = 0$  was feasible, whereas if the objective is to find a single optimal solution to (1),  $b_0$  is set equal to  $\bar{w}^\beta b^\beta - \delta$ .† A suitable value for  $\delta$  is the greatest common divisor of the

† By varying the value of  $b_0$  and introducing the constraint  $wb \geq b_0'$ , the algorithm will of course accomplish the objective of finding all feasible solutions with an objective function value lying within the range  $b_0' \leq wb \leq b_0$ .

nonzero  $b$ , when  $b$  is integer, and of course  $\delta = 1$  will work. When  $b$  is not integer some other value for  $\delta$  may need to be specified.

It is also to be noted that whenever  $b^\alpha \geq 0$ —as, for example, when (1) is in dual-feasible form with  $b$  nonnegative—the trial solution  $w^\alpha = 0$  is locally optimal whenever it is feasible. Thus, no better solution may be found along the present branch of the tree and the appropriate next step is the backtracking step, provided the objective is to find a single optimal solution to (1). This is merely a special case of a more general prescription for terminating investigation along a particular branch of the solution tree. It serves to illustrate, however, that whenever certain facts are known about the problem structure in advance, the tests to follow may be simplified to yield the desired information at a reduced computational cost.†

In developing the tests of this section, we will adhere to the following conventions and definitions.

1. All problem constraints, including the objective function constraint  $w(-b) \geq -b_0$ , will be represented in the form of the  $s$ -constraint  $wa \geq c_0$ .

2. All tests will be defined relative only to those constraints that are not eliminated as nonbinding, and relative only to those variables that are not currently assigned a specific value. This means that the constraint  $wa \geq c_0$  is a shorthand representation for  $w^\alpha a^\alpha \geq c_0'$ , where the latter is defined as for problem (3) above. These restrictions apply also to the range of definition of the special summation symbols to follow.

3.  $\sum^+ a_i$  will denote the sum of all positive  $a_i$  in  $a$ , and  $\sum^- a_i$  will denote the sum of all negative  $a_i$  in  $a$ .

4.  $\sum_H a_i$  will denote the sum of the  $H$  largest  $a_i$  in  $a$ , and  $\underline{\sum}_H a_i$  will denote the sum of the  $H$  smallest  $a_i$  in  $a$ .

5.  $\sum_{M,N} a_i$  will denote the maximum sum of the  $a_i$  such that there are at least  $M$  and no more than  $N$  terms in the sum. Thus

$$\sum_{M,N} a_i = \begin{cases} \sum_N^+ a_i & \text{if there are more than } N \text{ positive } a_i, \\ \sum_M^+ a_i & \text{if there are at least } M \text{ but no more than } N \text{ positive } a_i, \\ \sum_M^+ a_i & \text{if there are less than } M \text{ positive } a_i. \end{cases}$$

6.  $\underline{\sum}_{M,N} a_i$  will denote the minimum sum of the  $a_i$  such that there are at least  $M$  and no more than  $N$  terms in the sum. Thus

$$\underline{\sum}_{M,N} a_i = \begin{cases} \sum_N^- a_i & \text{if there are more than } N \text{ negative } a_i, \\ \sum_M^- a_i & \text{if there are at least } M \text{ but no more than } N \text{ negative } a_i, \\ \sum_M^- a_i & \text{if there are less than } M \text{ negative } a_i. \end{cases}$$

To develop several of the tests to follow, we will use the foregoing definitions to determine permissible values for two parameters,  $L$  and  $U$ .

† It is of course always possible to make  $b \geq 0$  by substituting  $v_i = 1 = w_i$  for  $b_i < 0$ . However, other structures may have a significantly greater influence on computation than dual-feasibility, as will be seen.

### Determination of $L$

$L$  denotes a lower bound on the number of free variables that must be set equal to one in order to obtain a feasible-optimal solution along the current branch of the solution tree, provided such a solution exists. If  $c_0 \leq 0$  for each constraint, then  $L=0$ . When  $c_0 > 0$ , a permissible value for  $L$  is given by

$$\sum_{L-1} a_i < c_0 \quad \text{and} \quad \sum_L a_i \geq c_0,$$

where we define  $\sum_0 a_i = 0$ . If there is no  $L$  that satisfies these conditions for  $c_0 > 0$ , then an optimal solution does not exist along the current branch.

*Level A.* Construct an  $s$ -constraint by which to determine  $L$ , using the objective function. Minimize  $\sum w_i$ . By Theorem 6A, the above determination of  $L$  optimizes this objective function relative to any  $s$ -constraint.

*Level B.* Determine a value for  $L$  from each of the problem constraints, including the objective function constraint and  $s$ -constraints, and select the largest value obtained.

*Level C.* Determine a value for  $L$  as in Level B from a restricted subset of the problem constraints—as, for example, from a single  $s$ -constraint already created.

*Level D.* Let  $L=0$ .

### Determination of $U$

$U$  denotes an upper bound on the number of free variables that may be set equal to one. Thus a permissible value for  $U$  is given by

$$\sum_U a_i \geq c_0 \quad \text{and} \quad \sum_{U+1} a_i < c_0,$$

provided it is possible to find a value for  $U$  that satisfies the above relations. If the first relation cannot be satisfied, then there is no optimal solution along the current branch of the tree. If the second relation cannot be satisfied,  $U$  is set equal to the number of free variables.

*Level A.* Construct an  $s$ -constraint to determine  $U$ , using the objective function. Minimize  $\sum -w_i$ . The given value of  $U$  is optimal by Theorem 6B.

*Level B.* Determine a value for  $U$  from each of the problem constraints, and select the smallest value obtained.

*Level C.* Determine a value for  $U$  as in Level B from a restricted subset of the problem constraints.

*Level D.* Let  $U$  equal the number of free variables.

The use of the following tests is based on the description of the algorithm in the second section. Thus, when local termination (backtracking) is prescribed, the solution process proceeds to the backtracking

and global termination step. When a variable  $w_i$  is prescribed to equal 1 (or 0), the term  $i$  (or  $\bar{i}$ ) is underlined and added to the solution sequence.

Except for Test 1, which is based only on  $L$  and  $U$ , the tests may generally be applied to all problem constraints, the  $s$ -constraints, and the objective function constraint—or to some selected subset of these constraints. The objective function constraint of course does not exist until a value is determined for  $b_0$ , but thereafter no solution is regarded feasible unless it satisfies this constraint. It is assumed that the value of  $b_0$  is adjusted appropriately whenever the tests uncover a feasible trial solution.

Interrelations between the tests are discussed in a series of notes to follow, which also suggest how the tests may be used to advantage. The prescriptions of the tests, and assertions about the tests found in the notes, may be seen to be valid by referring to definitions of terms given earlier. In those instances in which the validity of a remark is not immediate, explanation is added.

*Test 1.* If  $L > U$ , if  $U = 0$ , or if the determination of  $L$  or  $U$  indicates the absence of an optimal solution along the current branch of the solution tree, terminate investigation of the current branch. (When  $U = 0$  the trial solution is first checked for feasibility.) If  $L$  equals the number of free variables, set  $w_i = 1$  for all free  $w_i$ , provided the resulting solution is feasible, and then backtrack.

NOTE 1. Test 1 should always be performed immediately after any determination of  $L$  and  $U$  in order to avoid the possibility of applying subsequent tests unnecessarily. If  $L$  and  $U$  are obtained by a Level  $D$  determination, Test 1 reduces simply to checking whether any free variables remain, and terminating locally if not.

*Test 2.* If  $\sum_{L,U} a_i < c_0$ , there exists no feasible solution along the current branch of the solution tree. Terminate investigation of this branch and backtrack.

*Test 3.* If  $a_k > \sum_{L+1,U+1} a_i - c_0$ , where  $a_k$  is a term in the summation  $\sum_{L,U} a_i$ , then set  $w_k = 1$ .

NOTE 2. Test 3 is equivalent to setting  $w_k = 1$  when the assumption  $w_k = 0$  yields  $\sum_{L,U} a_i < c_0$ . It may be observed that  $\sum_{L+1,U+1} a_i - c_0$  must be computed only once for a given constraint, and thereafter acts as a screen to determine which variables may be forced equal to one. Thus if the relation of Test 3 holds for one  $a_k$ , it holds for all  $a_i \geq a_k$ , and conversely, if it fails to hold for some  $a_k$ , it will fail to hold for all  $a_i \leq a_k$ . In view of this fact, and to facilitate computation of  $\sum_{L,U} a_i$  generally, it may be desirable to rank the variables in terms of the magnitudes of their coefficients in  $wa \geq c_0$ , and to record this ranking before starting to solve the problem. If carried out for each constraint such a ranking procedure

would increase memory requirements to the extent of adding a second 'A matrix,' though if the constraints can be arranged according to similarities of ranking—or in terms of systematic differences in ranking—the storage requirements might be kept substantially lower. When storage limitations are not acute, computational gains to be derived by this procedure would unquestionably justify its use.

*Test 4.* If  $a_k < c_0 - \sum_{L-1, U-1} a_i$ , where  $a_k$  is not a term in  $\sum_{L, U} a_i$ , then set  $w_k = 0$ .

NOTE 3. Test 4 sets  $w_k = 0$  if  $\sum_{L, U} a_i < c_0$  is implied by  $w_k = 1$ . Note 2 concerning the statement of Test 3 applies also to Test 4.

NOTE 4. The stipulation that  $a_k$  is a term in  $\sum_{L, U} a_i$  in Test 3, and that  $a_k$  is not a term in that summation in Test 4, avoids unnecessary testing under the assumption that Test 2 has been applied beforehand. If Test 1 precedes Tests 2, 3, and 4, then there is no danger that the special summation symbols appearing in the latter tests will be undefined. However, the convention that all summations otherwise undefined must equal zero will produce valid tests in any event. Because of the overlap of computation involved in determining  $\sum_{L, U} a_i$ ,  $\sum_{L+1, U+1} a_i$ , and  $\sum_{L-1, U-1} a_i$ , as well as for the other reasons mentioned, it is generally desirable to apply Tests 2, 3, and 4 (in order) to a single constraint before applying any of them to another constraint.

NOTE 5. If  $L$  and  $U$  are not redetermined after testing a constraint that sets  $w_k = 1$  by Test 3 (or Test 6 following), then  $L$  and  $U$  should be reduced by the number of variables set equal to 1, though  $L$  need not be reduced below zero, and  $U$  is alternately bounded from above by the number of free variables. Adjusted values of  $L$  and  $U$  may of course serve as bounds on values of  $L$  and  $U$  obtained by redetermination.

NOTE 6. While Test 1 should precede Tests 2, 3, and 4 immediately after any determination of  $L$  and  $U$ , it should also again be applied whenever the values of  $L$  and  $U$  are adjusted as specified in Note 5. Although such an application will be useless until either  $L$  is reduced to 0 or  $U$  becomes equal to the number of free variables, it is computationally about as expedient to apply Test 1 as it is to check whether it ought to be applied.

*Test 5.* Let  $Z$  denote the value of the objective function that is produced by the optimal fractional solution to (2), or by the optimal integer solution to (2) if Theorem 6A or 6B is applicable. If  $Z > b_0$ , or if (2) has no feasible solution (as identified by Theorem 5A or 5B), terminate investigation of the present branch of the solution tree.

*Test 6.* Let  $Z_k$  denote  $Z$  obtained under the assumption that  $w_k = 0$ , where  $w_k \neq 0$  in the specified solution to (2) obtained without this assumption. If  $Z_k > b_0$ , or if (2) has no feasible solution when  $w_k = 0$ , set  $w_k = 1$ .

NOTE 7. It is evident that if Test 6 sets  $w_k = 1$ , then it will also set  $w_i = 1$  for any  $i$  for which  $a_i \geq a_k$  and  $b_i \leq b_k$ . Similarly, if Test 6 does not set  $w_k = 1$ , then neither will it set  $w_i = 1$  for any  $i$  for which  $a_i \leq a_k$  and  $b_i \geq b_k$ . Note that the auxiliary indexing defined in the third section may be used as a guide to determining whether such relations hold in the cases covered by Theorems 5A and 5B. For example, if  $a, b > 0$ , and  $w_k = 1$  by Test 6, then  $w_{i_p} = 1$  for any  $p < k$  for which  $a_{i_p} \geq a_{i_k}$ .

Test 7. Let  $Z^k$  denote  $Z$  obtained under the assumption that  $w_k = 1$ , where  $w_k \neq 1$  in the solution obtained without this assumption. If  $Z^k > b_0$ , or if (2) has no feasible solution when  $w_k = 1$ , set  $w_k = 0$ .

NOTE 8. If  $w_k$  is set equal to 0 by Test 7, then  $w_i = 0$  for any  $i$  such that  $a_i \leq a_k$  and  $b_i \geq b_k$ . If Test 7 does not set  $w_k = 0$ , then neither will it set  $w_i = 0$  for any  $i$  for which  $a_i \geq a_k$  and  $b_i \leq b_k$ . The auxiliary indexing may be used as a guide to these relations in the manner suggested in Note 7.

NOTE 9. Test 5 should be applied prior to Tests 6 and 7 relative to any given constraint for the following reasons: (a) Test 5 may lead to termination, making the other tests unnecessary, and (b) information about the values of the variables that optimize (2)—obtained by Test 5—is used directly in Tests 6 and 7. It should also be noted that there is an overlap in computation of  $Z$ ,  $Z_k$ , and  $Z^k$  that may be exploited by applying Tests 5, 6, and 7 to a given constraint without intervening testing.

NOTE 10. The variables  $w_i$  of Theorem 5A and  $w_i$  of Theorem 5B have the unique property that they may be set equal to 1 by Test 6 and also set equal to 0 by Test 7. In either case, after recording the assigned value, Tests 5, 6, and 7 should be reapplied. Thus it is expedient to apply Test 6 and Test 7 immediately to  $w_i$  (or  $w_i$ ) after Test 5, and if results are obtained, to return at once to Test 5. If no results are obtained at Test 6 or 7 for  $w_i$  (or  $w_i$ ), then in view of Notes 7 and 8 the most efficient procedure would be to apply Test 6 to all remaining variables over which it has jurisdiction, and then to apply Test 7 to those variables over which it has jurisdiction.

Test 8. If  $\sum_{L,U} a_i \geq c_0$ , then the constraint  $wa \geq c_0$  is nonbinding, and is removed from the problem until the last nonunderlined term in the solution sequence becomes underlined at the backtracking step.

NOTE 11. A constraint that is found to be nonbinding is handled by the same rule as a variable that is assigned a specific value by one of the preceding tests. In fact, when constraint 4 (say) is found to be nonbinding, the term  $C_4$  could be added to the solution sequence and underlined, exactly as 4 would be added and underlined when Test 3 sets  $w_4 = 1$ . Since constraint 4 is rendered nonbinding by the solution sequence preceding the term  $C_4$ , it will remain nonbinding until that sequence is changed, hence the reason for the rule of Test 8. Because of the special status of the ob-

jective function constraint and the  $s$ -constraints, it is usually desirable to skip them when applying this test.

NOTE 12. Though we do not bother to employ them here, other tests based upon the theorems of the third section may also be developed. Given any two constraints  $wa^1 \geq c_0^1$  and  $wa^2 \geq c_0^2$ , we may create the problems, minimize  $wa^1$ , subject to  $wa^2 \geq c_0^2$ , and minimize  $w(-a^1)$ , subject to  $wa^2 \geq c_0^2$ . If the solution to the former problem yields a value for  $wa^1$  that is as large as  $c_0^1$ , then  $wa^1 \geq c_0^1$  may be removed as nonbinding, while if the solution to the latter problem yields a value for  $w(-a^1)$  that is greater than  $-c_0^1$ , then problem (1) has no feasible solution along the current branch of the tree, and termination is prescribed. The latter problem may also be used to force some of the problem variables to assume specific values in the manner of Tests 6 and 7. It is to be noted that the two foregoing problems can also be used to establish additional constraints to be added to problem (1) if this is desired. Thus, if  $wa^2 \geq c_0^2$  is given or implied by problem (1), then a permissible range of values for  $c_0^1$  can be determined directly as above for any arbitrary vector  $a^1$ . For the special case in which  $a^1$  consists of all 1's, the indicated procedure is precisely the one specified earlier for determining  $L$  and  $U$  (using Theorems 6A and 6B). Generalization of these tests to solving a linear or integer linear programming problem in more than one constraint throws away some of the computational advantages afforded by the theorems, but may have value in some applications.

### A SPECIFIC VERSION OF THE ALGORITHM

THERE ARE a number of factors that determine how the preceding tests should be used to produce the best results. We will return to such considerations later. First, however, three example problems will be solved with the version of the algorithm that produced the computational results reported in the first section. This version is summarized in Diagram 2, and discussed in the guide that follows.

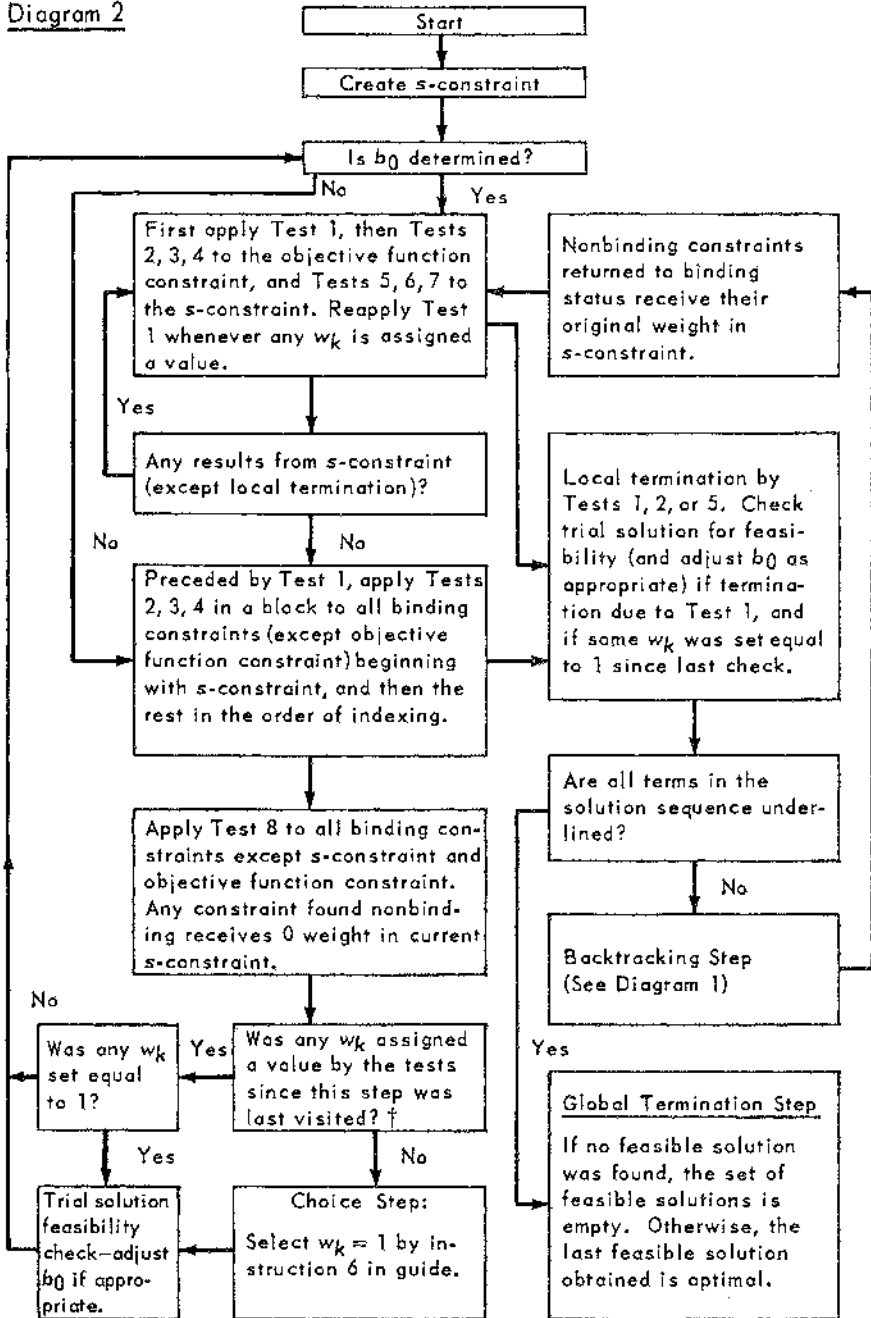
#### Guide to Diagram 2

1. A single  $s$ -constraint is used, determined initially by Method  $B$  of the third section, and selecting  $\epsilon$  from the range  $0 < \epsilon \leq 0.2$ . (We allow  $\epsilon$  to vary for convenience in solving example problems by hand so that the multiple  $f/g + \epsilon$  can be expressed in tenths, and, if possible, as an integer.) To avoid excessive computation, the third constraint generated by Method  $B$  is automatically selected if the first two do not qualify.

2. To keep the  $s$ -constraint current, the weight of any constraint composing it is allowed to go to zero when that constraint is found to be nonbinding by Test 8. When the algorithm backtracks to a point where the nonbinding constraint becomes binding, the constraint is once again assigned its original weight in the  $s$ -constraint.

3. The tests are based on a Level  $D$  determination of  $L$  and  $U$ . Thus Test 1 becomes simply a check to see whether any free variables remain, and the expression

Diagram 2



† When this step is first visited, the reference to the "last time visited" is of course ignored.

$\sum_{L,U} a_i$  in Test 2 reduces to  $\sum^+ a_i$ . The other tests are similarly simplified. Note also that Test 3 may be bypassed for any constraint in which all  $a_i$  are non-positive, and Test 4 may be bypassed for any constraint in which all  $a_i$  are non-negative.

4. If any  $w_k$  is assigned a specific value by one of the tests, this fact is immediately recorded (and the  $c$  vector adjusted for  $w_k=1$ ) to make this information available in testing the constraints to follow.

5. Once the tests have been applied to all constraints, a check is made to determine if any new variables have been assigned specific values. If so, the tests are repeated. It is to be noted, however, that the repeat testing need not be applied to those constraints that were tested (other than with Test 8) subsequent to the last point at which some  $w_k$  was set equal to 0 or 1.

6. At the choice step of the algorithm, one of the free variables is selected to equal 1 according to criteria derived from the auxiliary indexing of the  $s$ -constraint. So that these selections may cover all circumstances, we extend the auxiliary indexing as follows.

A. For  $b > 0$ : Define the auxiliary indexing relative to those variables for which  $b_i$  and  $a_i > 0$  exactly as in Theorem 5A, so that Tests 5, 6, and 7 may be applied without alteration. The extended indexing is then defined so that

- (i) all nonpositive  $a_i$  are indexed higher than any positive  $a_i$ ;
- (ii)  $b_{i_p} - a_{i_p} > b_{i_q} - a_{i_q}$  implies  $p > q$  for  $a_{i_p}, a_{i_q} \leq 0$ .

Choice: Set  $w_{i_p} = 1$ , where  $p$  is the least auxiliary subscript belonging to a free variable.

B. For  $b < 0$ : Define the auxiliary indexing relative to those variables for which  $b_i$  and  $a_i < 0$  as in Theorem 5A. Then extend the indexing so that

- (i) all nonnegative  $a_i$  are indexed higher than any negative  $a_i$ ;
- (ii)  $a_{i_p} - b_{i_p} > a_{i_q} - b_{i_q}$  implies  $p > q$  for  $a_{i_p}, a_{i_q} \geq 0$ .

Choice: Set  $w_{i_p} = 1$ , where  $p$  is the largest auxiliary subscript belonging to a free variable.

We will only need the above two cases for the example problems to follow. However, we include case C below for the sake of completeness.

C. For  $b$  otherwise: Define a single auxiliary indexing as in Theorem 5A over all variables for which  $b_i, a_i > 0$  and  $b_i, a_i < 0$ . Note that for these variables this indexing will remain unchanged when (2a) is used in Tests 5, 6, and 7. Extend the auxiliary indexing so that

- (i) all other variables will be indexed higher than those for which  $b_i, a_i > 0$  and  $b_i, a_i < 0$ ;
- (ii) variables for which  $a_i \geq 0, b_i \leq 0$  are indexed higher than those for which  $a_i \leq 0, b_i \geq 0$  ( $a_i = b_i = 0$  may be put in either category);

(iii) the indexing for  $a_i \leq 0, b_i \geq 0$  is given by (ii) of Case A, and the indexing for  $a_i \geq 0, b_i \leq 0$  is given by (ii) of Case B.

Choice: (a) Set  $w_{i_p} = 1$  for the largest  $p$  for which  $a_{i_p} \geq 0$  and  $b_{i_p} \leq 0$ , if any such  $p$  exists.

(b) If (a) cannot be executed, and if  $c_0 \geq 0$ , set  $w_{i_p} = 1$  for the smallest  $p$  such that  $a_{i_p}, b_{i_p} > 0$ , if such a  $p$  exists.

(c) If (a) cannot be executed, and if  $c_0 < 0$ , set  $w_{i_p} = 1$  for the largest  $p$  such that  $a_{i_p}, b_{i_p} < 0$ , if such a  $p$  exists.

(d) If none of the foregoing instructions can be carried out, set  $w_{i_p} = 1$  for the smallest  $p$  such that  $a_{i_p} \leq 0$  and  $b_{i_p} \geq 0$ .

It may readily be seen that the algorithm of Diagram 2 (elaborated upon in the preceding guide) fits the general pattern outlined in Diagram 1 of the second section, so that the version of the algorithm employed here must find an optimal solution if one exists. To apply the algorithm, we will use the format exemplified in Table II.

TABLE II  
FORMAT FOR THE ALGORITHM

	$-b$	$A$	$S_0$	$S_2$	
	—	$c$		—	
(1)	—	$c^1$	$s^1$	—	$i_1$
(2)	—	$c^2$	$s^2$	—	$i_1, i_2, \bar{i}_3, \bar{i}_4$
(3)	$-b_0^3$	$c^3$	—	$s^3$	$i_1, i_2, \bar{i}_3, \bar{i}_4, i_5, i_6$ $F$
(4)	$-b_0^4$	$c^4$	—	$s^4$	$i_1, i_2, \bar{i}_3, \bar{i}_4, i_5, i_6$ $FT$
(5)	$-b_0^5$	$c^5$	—	$s^5$	$i_1, i_2, \bar{i}_3, \bar{i}_4, \bar{i}_5, \bar{i}_7$ $T$
(6)	$b_0^6$	$c^6$	$s^6$	—	$i_1, \bar{i}_2, i_3, \bar{i}_3$
	$\vdots$	$\vdots$	$\vdots$		$\dots$
(k)	$-b_0^k$	$c^k$	$s^k$	—	$\bar{i}_1, \bar{i}_2, \bar{i}_3$ $TT$

Table II is explained as follows. The terms  $i_1, i_2$ , etc., to the lower right of the tableau represent the solution sequence described in the second section. The rows directly beneath the tableau represent the current  $c$  vector. Thus  $c^1$  is equal to the  $c$  vector less row  $i_1$  of the  $A$  matrix (corresponding to  $w_{i_1} = 1$ ),  $c^2$  is equal to  $c^1$  less rows  $i_2$  and  $i_4$  of the  $A$  matrix (corresponding to  $w_{i_1} = w_{i_2} = w_{i_4} = 1$  and  $w_{i_3} = 0$ ), and so forth. Each row beneath the tableau is produced by a 'cycle' of the algorithm, which is completed when (a) a new variable is assigned a value at the choice step and the solution sequence is thereupon updated to include all terms that are specified to be added by the tests, (b) a new feasible solution is obtained, or (c) a local termination is executed. The symbols  $F$  and  $T$  respectively denote feasibility and termination. A solution with no symbol beside it is infeasible, but of course nonterminal.† The symbol  $TT$  represents global termination.

† When we speak of feasibility of an incomplete 0-1 assignment we refer to the feasibility of the corresponding trial solution.

The next to the last column of the tableau is the original  $s$ -constraint, denoted  $S_0$ . The last column is a second  $s$ -constraint,  $S_2$ , which is produced from  $S_1$  by assigning zero weight to constraints determined nonbinding upon obtaining  $c^2$ . To clarify how these constraints are used, and to explain the remainder of the format, we will trace through the process of solving the hypothetical problem of Table II.

On line 1,  $w_{i_1}$  is selected for the first variable to equal one, producing the vector  $c^1$ , but no entry is indicated beneath the  $-b$  vector since a value for  $b_0$  has not yet been determined. Similarly,  $w_{i_2}$  is next selected to equal

Example Problem 1

	1	1	1	1	1	1	1	$S_0$	$S_1$	$S_2$	
-6	-3	0	-2	5	0	0	7	7 (7)	7 (6)	10 (4)	
-5	8	-1	2	-3	0	8	-4	10 (6)	11 (4)	3 (7)	
-1	8	5	1	-1	2	0	-3	12 (1)	—	—	
-7	-1	0	0	0	3	6	6	14 (5)	14 (5)	15 (3)	
-2	0	5	-1	0	0	6	1	11 (3)	6 (3)	6 (2)	
-4	0	-1	0	0	4	-5	0	-2 (10)	-1 (9)	-1 (8)	
-3	0	-1	0	-1	-1	0	5	2 (8)	3 (7)	3 (6)	
-1	-1	-1	2	-1	5	-2	1	3 (4)	4 (2)	5 (1)	
-5	6	0	0	1	2	12	8	29 (2)	29 (1)	—	
-3	-2	0	-1	1	0	4	0	2 (9)	2 (8)	4 (5)	
—	2	1	3	-1	4	7	5	21	—	—	
(1)	-6	<u>-4</u>	2	0	2	7	8	11	13	—	3
(2)	-12	—	2	-1	2	-5	-3	—	-16	-5	3, 9
(3)	1	—	0	0	-3	-3	-4	—	—	-10	3, 9, 8 FT
(4)	0	—	—	—	—	—	—	—	—	—	3, 9, $\bar{2}$ T
(5)	-5	-6	—	—	—	—	—	—	<u>2</u>	—	3, $\bar{0}$ , $\bar{1}$ , $\bar{4}$ , $\bar{2}$ T
(6)	-1	-4	1	3	-2	2	-5	-8	—	—	3, $\bar{4}$ , $\bar{9}$ , $\bar{1}$ , $\bar{2}$ , $\bar{5}$ , $\bar{6}$ , $\bar{7}$ , $\bar{10}$ TT

1 on line 2. Here, however, the tests of the algorithm indicate that  $w_{i_3}=0$  and  $w_{i_4}=1$  are implied by  $w_{i_1}=w_{i_2}=1$ ; hence the terms  $\bar{i}_3$  and  $\bar{i}_4$  are added and underlined. At this point, one or more of the constraints of the problem become nonbinding; hence their weights in  $S_0$  are reduced to 0, producing  $S_2$ . The scalars  $s^1, s^2, \dots, s^6$  indicate by their position which  $s$ -constraint is employed at various stages of the algorithm. By Note 11 following Test 8, we know that the constraints that become nonbinding for the solution sequence of line 2 will necessarily remain nonbinding until  $\bar{i}_2$  is replaced by  $\bar{i}_2$ , so that  $S_2$  will take precedence over  $S_0$  until line 6 is reached, after which  $S_2$  may be dropped. Of course, the determination of other nonbinding constraints may produce  $s$ -constraints (not shown) to supersede  $S_0$  and  $S_2$  at intermediate points in the solution process.

On line 3, the choice of  $w_{i_3}=1$  produces a feasible trial solution, as indicated by the symbol  $F$ . Thus a value is determined for  $-b_0$ , and is

entered beneath the  $-b$  vector. Since line 3 represents the *current* bottom row of the tableau,  $-b_0^3=1$  (under the assumption that the  $b$  vector is integer). On line 4,  $w_{i_4}$  is set equal to one by the tests, evidently as a result of the restriction imposed by  $-b_0^3$ , and the tests thereupon register a local termination for which the trial solution is feasible (denoted by *FT*). The value of  $-b_0^4$  is thus readjusted so that  $-b_0^4=1$ .

On line 5,  $\bar{i}_5$  is forced to replace  $i_5$  at the backtracking step, and thereupon the tests set  $w_{i_7}=0$ . On line 6,  $\bar{i}_2$  replaces  $i_2$  at the backtracking step, and the term  $i_5$  is added to indicate that  $w_{i_5}$  is selected to equal one. When this is accomplished the tests set  $w_{i_3}=0$ , so that  $\bar{i}_3$  is added and underlined. It may be noted that  $c^6$  is equal to  $c^1$  minus row  $i_5$  of the  $A$  matrix. In this fashion earlier rows may be used to shortcut computation of later rows. Rows lying between line 1 and line 6 will of course never be used in this manner since they all contain the sequence  $i_1, i_2$  which is now replaced by  $\bar{i}_1, \bar{i}_2$ . The remaining steps add nothing new to the format, and the algorithm terminates as on line  $k$ .

For Example Problem 1 we have listed the weights of the problem constraints composing the original  $s$ -constraint  $S_0$  on the top row above the tableau. Thus all constraints are assigned a unit weight. This is because the second  $s$ -constraint generated by Method B produces a lower-valued objective function value with Approximation 1 of the third section than the first  $s$ -constraint; hence the first ( $S_0$ ) is the one employed. The numbers in parentheses beside the components of the  $s$ -constraints  $S_0, S_1$ , and  $S_2$  indicate the auxiliary indexing, which is used both in Tests 5, 6, 7, and in determining the variable to set equal to one on the choice step.

*Line 1.* Applying the tests to the problem constraints does not disclose any variables to be set equal to zero or one, nor does Test 8 find any non-binding constraints. Thus,  $w_3$  is selected to equal one, since it has the least auxiliary index of the free variables in  $S_0$ . The tests are again applied, after checking for feasibility, and constraint 2 is determined non-binding. Thus the  $-4$  beneath constraint 2 on line 1 is underlined, and this constraint will not be considered again until 3 is replaced by  $\bar{3}$  in the solution sequence. The weight of constraint 2 is dropped to 0 in the  $s$ -constraint, producing  $S_1$ , which now takes precedence over  $S_0$ .

*Line 2.*  $w_3$  is selected to equal one since it has the lowest rank of the free variables in  $S_1$ . Note that we have not bothered to compute the coefficient of  $w_3$  in  $S_1$ , since  $w_3$  cannot be free at any time during which  $S_1$  has precedence. The tests of the algorithm do not assign any other variables a specific value, but constraint 1 is registered nonbinding, yielding  $S_2$ .

*Line 3.* In  $S_2$ ,  $w_8$  has the lowest rank of the free variables, and  $w_8$  is set equal to 1. The trial solution is feasible; hence  $-b_0$  is assigned a value of 1. Test 2 is immediately failed for the objective function constraint, so that the process terminates locally.

*Line 4.* The term 8 is replaced by  $\bar{8}$  at the backtracking step; initially this line begins the same as line 2, except that  $-b_0=0$ . However, Test 2 is again failed for the objective function constraint, so that the process terminates without selecting a new variable to equal one. Since it is unnecessary to compute anything new for this line other than  $-b_0=0$ , it has been left blank.

*Line 5.* The term 9 is replaced by  $\bar{9}$  at the backtracking step, and  $S_1$  now takes precedence over  $S_2$ , which may be dropped. Line 5 can now be derived from line 1—lines 2, 3, and 4 of course will not serve in this fashion

Example Problem 2

	3.8	1	1	1	1.3	1.3	$S_0$	$S_2$	
6	-3	-1	0	-1	-1	0	-14.7 (2)	-14.7 (1)	
4	0	-3	-2	2	-1	-1	-5.6 (6)	—	
9	-3	-2	-1	2	-2	0	-15. (5)	-14. (3)	
3	0	-2	0	1	-1	-3	-6.2 (4)	—	
4	-2	-1	-2	0	-1	-1	-13.2 (1)	—	
5	-3	0	-2	3	-1	0	-11.7 (3)	-9.7 (2)	
4	0	-1	-1	1	-1	-2	-4.9 (7)	—	
8	-1	2	1	0	-2	-2	-6. (8)	—	
—	-5	-3	-4	4	-5	-3	-32.4	—	
(1)	—	-4	-5	-5	4	-3	-26.4	—	$8, \bar{2}, \bar{3}$
(2)	—	-4	-2	-3	2	-2	-20.8	-17.8	$8, \bar{2}, \bar{4}, 2, \bar{5}$
(3)	1	-1	0	—	0	0	—	-3.8	$8, \bar{2}, \bar{4}, 2, \bar{5}, 3, \bar{1}, \bar{6}$ FT
(4)	10	2	—	—	—	—	—	<u>6.6</u>	$8, \bar{2}, \bar{4}, 2, \bar{5}, \bar{3}, \bar{1}, \bar{6}$ T
(5)	14	—	—	—	—	—	-26.4	—	$8, \bar{2}, \bar{4}, \bar{2}$ T
(6)	22	—	—	—	—	—	-32.4	—	$\bar{8}$ TT

as the basis for any subsequent lines. Test 4 applied to the objective function constraint yields  $w_1$  and  $w_4=0$ . The constraint  $S_1$  passes Test 5, but  $w_2$  is set equal to 1 by Test 6. Since  $w_2$  is  $w_{i_r}$ , this is the situation referred to in Note 10 of the preceding section, so that Test 5 is repeated, taking  $w_2=1$  into consideration. This time Test 5 is failed, so that the process terminates. Since it was unnecessary to adjust any of the bottom row except for  $S_1$  and the objective function constraint, the remaining entries on line 5 are left blank.

*Line 6.* Term 3 is replaced by  $\bar{3}$  at the backtracking step, and  $S_0$  is reinstated over  $S_1$ . Test 4 applied to the objective function constraint sets  $w_4=0$ , and Test 6 applied to  $S_0$  sets  $w_3=1$ . Here  $w_3=w_{i_r}$ , so that Test 5 is repeated, but it is passed. Test 7, however, forces all free variables to 0 except  $w_3$ . Since results were obtained from the  $s$ -constraint, the objective function constraint is retested, but nothing new is obtained. Test 2 is failed for constraint 2, and global termination ensues. The solution of line 3— $w=(0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0)$ —is the optimal solution.

Example Problems 2 and 3 are explained in less detail than Example Problem 1.

*Example Problem 2.* The  $s$ -constraint  $S_0$  is the third created by Method B.

*Line 1.*  $w_8=1$  by choice (largest auxiliary index),  $\bar{7}$ ,  $\bar{4}$  from constraint 6 (Test 4).

*Line 2.*  $w_2=1$  by choice,  $\bar{5}$  from constraint 6. Constraints 3 and 6 become nonbinding, yielding  $S_2$ .

*Line 3.*  $w_3=1$  by choice. Trial solution feasible.  $\bar{1}$ ,  $\bar{6}$  from  $S_2$  (Test 4). Terminate by Test 1—no more free variables.

Example Problem 3

	4	1	1	1	4	1	1	$S_0$	$S_1$	$S_2$
7	-4	0	0	1	-2	-5	2	-26 (5)	—	-26 (4)
3	-1	-2	-1	0	-1	-4	1	-14 (4)	—	-13 (3)
5	0	-2	0	0	-2	0	2	-8 (6)	—	-8 (5)
9	-3	0	-1	1	-1	0	1	-15 (7)	—	—
2	-2	1	1	0	-1	-6	0	-16 (1)	-18 (1)	-17 (1)
6	0	0	-2	1	-2	0	3	-6 (8)	—	—
1	0	-2	-2	1	-1	2	-1	-6 (3)	—	-4 (2)
4	-4	1	0	0	-3	0	3	-24 (2)	—	—
—	-6	-3	-3	2	-4	-8	6	-46	—	—
(1)	—	-6	-3	-1	1	-2	-8	-40	—	—
(2)	—	-3	-3	0	0	-1	-8	-25	—	—
(3)	—	-6	-1	-1	1	0	-8	-32	—	—
(4)	—	-1	-1	0	0	1	1	0	1	—
(5)	—	-3	-1	0	0	-2	-10	-14	—	—
(6)	—	-2	-4	-3	2	-1	-8	-22	—	-19

$\bar{6}, \bar{8}$   
 $\bar{6}, \bar{8}, \bar{4}, \bar{1}, \bar{2}, \bar{3}, \bar{7}$  T  
 $\bar{6}, \bar{8}, \bar{4}, \bar{3}, \bar{1}, \bar{2}, \bar{5}, \bar{7}$   
 $\bar{6}, \bar{8}, \bar{4}, \bar{3}, \bar{1}, \bar{2}, \bar{7}$  T  
 $\bar{5}, \bar{4}, \bar{1}, \bar{8}, \bar{7}$  T  
 $\bar{5}, \bar{4}, \bar{8}, \bar{1}$  TT

*Line 4.*  $\bar{3}$  replaces 3 at the backtracking step,  $\bar{1}$ ,  $\bar{6}$  by objective function constraint (Test 3). Terminate by Test 1. Solution infeasible by  $S_2$ .

*Line 5.*  $\bar{2}$  replaces 2 at backtracking step, constraints 3, 6, and  $S_0$  are reinstated. Terminate by Test 5 on  $S_0$ .

*Line 6.*  $\bar{8}$  replaces 8 at backtracking step. Terminate by Test 5 on  $S_0$ . It is unnecessary to generate the complete bottom row for lines 4, 5, and 6. Line 3 gives the optimal solution.

*Example Problem 3.* The  $s$ -constraint  $S_0$  is the third created by Method B.

*Line 1.* Choose  $w_4=1$  (largest auxiliary index).  $\bar{8}$  from constraint 5 (Test 4).

*Line 2.* Choose  $w_1=1$ .  $\bar{1}$  from  $S_0$  (Test 4),  $\bar{7}$  from constraint 3,  $\bar{2}$  from constraint 5. Terminate by constraint 7 (Test 2).

*Line 3.*  $\bar{4}$  replaces 4 at backtracking step.  $w_3=1$  by choice,  $\bar{1}$ ,  $\bar{2}$ ,  $\bar{5}$ ,  $\bar{7}$  from constraint 5 (Test 4). Terminate by Test 1.

*Line 4.*  $\bar{3}$  replaces 3 at backtracking step. 1, 2,  $\bar{7}$  from constraint 7 (Tests 3 and 4). Constraints 2, 3, 4, 7 become nonbinding, producing  $S_4$ . Terminate by  $S_4$  (Test 2).

*Line 5.*  $\bar{6}$  replaces 6 at backtracking step. Constraints 2, 3, 4, 7 return to binding status, and  $S_0$  is reinstated over  $S_4$ , which is dropped.  $w_4=1$  at choice step.  $\bar{1}$  from  $S_0$ ,  $\bar{8}$  from constraint 1, and  $\bar{7}$  from constraint 4. Terminate by constraint 7 (Test 2).

*Line 6.*  $\bar{4}$  replaces 4 at backtracking step.  $\bar{8}$  from constraint 7. Constraint 3 becomes nonbinding, producing  $S_6$ .  $\bar{1}$  by  $S_6$ . Terminate by constraint 4. The problem has no feasible solution.

The foregoing problems were constructed for illustrative purposes to be easy to solve, and were in some respects at a level of simplicity below that which the version of the algorithm in this section was designed to handle most efficiently. This does not mean that the examples are easy for other methods, for it is evident that attempting to solve them by hand with the better known integer programming algorithms would be a dismal, if not an overwhelming, chore.† The additive algorithm of Balas works better than most for these problems, the computational relation between the additive algorithm and the present method being as indicated in the first section. In fact, the particular version we have used here for illustrative purposes bears some strong resemblances to Balas' method in several respects. For example, the restricted forms of Tests 1 and 2 employed are likewise used in the additive algorithm, and Test 5 may be regarded as an extension of the ideas underlying Test 2 when this test is applied, as in Balas' method, to the constraint  $wb \geq b_0$ .‡ Nonetheless, in spite of the ease with which it handles the foregoing problems, the version of the algorithm employed here has a number of patent limitations that need not be tolerated when solving more complex problems with the use of a computer. We discuss some of these limitations, and ways of removing them, in the next section.

#### OTHER VERSIONS OF THE ALGORITHM

THERE ARE several considerations that exert a positive influence on the efficiency of the algorithm for more complex problems than solved above. We list these considerations below.§

† It is to be noted that times consumed in hand computation do not give an infallible guide to the relative efficiency of two algorithms, since computations that are onerous for humans may pose no comparable difficulties for high speed computers. On the other hand, there are some correspondences between the relative effort required by humans and computers for a number of common operations: e.g., multiplication and division take longer than addition and subtraction.

‡ Test 5 is still more closely linked to the work of GILMORE AND GOMORY<sup>10</sup> who employ an essentially equivalent test in solving the knapsack problem when the problem variables are not constrained by explicit upper bounds.

§ This list is not intended by any means to be exhaustive. For additional considerations of a related nature, see references 14 and 17.

1. The use of more advanced methods than Method B for creating  $s$ -constraints, and the use of more than one  $s$ -constraint in solving (1).
  2. The use of higher level determinations of  $L$  and  $U$ .
  3. The use of choice rules that include the possibility of setting a variable equal to zero at the choice step.
  4. The use of procedures for exploiting highly structured problems.
- Other considerations that have not been found as important for the problems so far examined, but that may in some cases influence performance as substantially as those listed above, are:
5. The use of a different enumerative substructure.
  6. The application of tests at earlier points in the solution sequence than the one currently visited (when a new value of  $b_0$  is obtained).

We will now discuss the preceding items in sequence.

Consideration 1 is extremely important. More powerful  $s$ -constraints than produced by Method B have been found to yield considerable computational gains for several problems beyond the level of complexity of the problems solved in the preceding section. Though it is beyond the scope of this paper to discuss them here, methods that produce such constraints are presented in reference 15. For some problems, of course, the individual constraints are independently restrictive enough that time devoted to constructing  $s$ -constraints by any method is of little value. Such was the case for the problem mentioned in the first section, which required 17 minutes to solve; without computing an  $s$ -constraint this problem would have required less than 7 minutes. However, when there is nothing to be gained from a problem by considering the effect of constraints in interaction (using an  $s$ -constraint), it may generally be expected that the problem will be simple enough that it may be solved easily in any case.

Consideration 2 is supported by the availability of  $s$ -constraints that can produce restrictive values of  $L$  and  $U$ . By using more than one  $s$ -constraint, it is sometimes possible to determine higher values for  $L$  and lower values for  $U$  than can be obtained from any individual constraint in the manner outlined in the fourth section. Such considerations are also discussed in reference 15. Generally speaking, it does not appear to be advantageous to redetermine  $L$  and  $U$  at exceedingly frequent intervals, but periodic determinations, with adjusted values maintained in the meanwhile, can be quite useful. The computational cost of such determinations is low, and if it is feasible to use a ranking procedure like that outlined in Note 2 of the fourth section, the tests based on high level determinations of  $L$  and  $U$  can be executed about as rapidly as those based on a level  $D$  determination—and they are unquestionably sharper.

Selecting a variable to equal zero at the choice step (consideration 3) may sometimes save a number of cycles of the algorithm and a great deal of testing. Such a situation occurs when the selected variable (call it  $w_k$ ) is not a particularly good variable to select to equal one—by the criteria

of choice used in the fourth section—and when (a) setting other variables equal to one tends to force  $w_k=0$ , or (b) setting  $w_k=0$  predisposes several other variables to be forced to equal one. Situation (a) often occurs when  $a_k$  is a large negative number in a constraint that is predominantly non-positive, and situation (b) often occurs when  $a_k$  is positive in a constraint for which  $L$  is determined to be somewhat greater than half the number of positive coefficients.† It would in many cases be a waste of time to search at each step of the algorithm to see if any  $w_k$  satisfied such criteria unless the likely variables were pinpointed ahead of time, thus avoiding an extensive search over a large number of variables and constraints.

To discuss consideration 4, we give two simple examples of 'highly structured' problems: (i) the problem contains a subset of constraints for which  $a_i \geq 0$  for all  $i$ , (ii) the problem contains a subset of constraints for which  $a_i \leq 0$  for all  $i$ . Both kinds of structures are often found in logic problems, capital budgeting problems, sequencing and scheduling problems, and many others. Constraints in the first category are nonbinding as soon as they are satisfied, and may be ignored thereafter until the solution process backtracks to a stage at which they were unsatisfied. For constraints in the second category, when the smallest negative coefficient is smaller than the current  $c_0$ , all variables associated with negative coefficients are forced to equal zero and the constraint becomes nonbinding. In the special case where every constraint is structured so that  $a_i = 0$  or 1 for all  $i$ , and  $c_0 = 1$  as well, a number of excellent shortcut techniques for ruling variables out of consideration have been developed by GORDON, *et al.* in reference 22.

The value of highly structured constraints is that they are highly restrictive—that is, they have an easily accessible information content that imposes strong limits on the set of optimal solutions. Restrictive constraints that cannot be recognized quite so readily by structure may be identified by trial testing. For example, if a constraint forces several variables to equal one or zero each time a handful of variables are assigned a specific value, then the constraint clearly belongs to the category about which we are speaking. Such constraints should be given priority in the algorithm so that they are tested before others, and then retested 'out of sequence' (as with the  $s$ -constraint and objective function constraint in the algorithm exemplified in the preceding section).

Employing a different enumeration procedure (consideration 5) leads generally back to the issue of balancing memory requirements against hoped-for computational gains. In the direction of greater complexity, the enumeration procedure employed by the algorithm might be patterned

† If  $L$  is initially found to be as large as half the number of problem variables, it will be generally worthwhile to replace all  $w_i$  by  $v_i = 1 - w_i$  before starting to solve the problem.

along the lines of the 'branch-and-bound' process of reference 24. In this case, if  $w_k$  was set equal to 1 at the choice step, a lower bound would immediately be computed for the objective function value implied by  $w_k=0$ . After a local termination, instead of returning to the closest preceding branch of the tree not yet investigated, the algorithm would return to the branch associated with the smallest of the computed lower bounds. (For the integer programming problem, it would probably be best to use some guide other than the lower bounds to determine the branch to return to, since the bounds tend to become more precise—and hence larger—the farther out one moves along the solution tree.) The chief difficulty of such a method of course lies in the need to keep track of all branches not yet explored and not otherwise ruled out of consideration, so that memory requirements could soon be pushed to their limits.

At the opposite pole, an even simpler enumeration procedure than used in this paper may be employed as follows. Index the problem variables in advance according to a priority system to be adhered to throughout the solution process. Then define the vector  $t=(t_1, t_2, \dots, t_m)$ , where  $t_i=0$  means  $w_i=0$  and the  $i$  term in the solution sequence is underlined,  $t_i=1$  means  $w_i=1$  and the  $i$  term in the solution sequence is nonunderlined, and  $t_i=2$  means that  $w_i$  is not currently assigned a value. Then the entire solution tree may be recorded by the three-valued components of  $t$  using the following rules: (a) at the choice step always select  $w_i=1$  for the least  $i$  such that  $t_i=2$ ; (b) apply Tests 3 and 6 only in conjunction with the backtracking step, so that the method continues to backtrack until it finds the largest  $i$  for which  $t_i=1$  and  $w_i$  is not compelled to equal 1; (c) Tests 4 and 7 are applied only to the least indexed  $w_i$  for which  $t_i=2$ ; (d) Tests 1, 2, 5, and 8 are applied as before.

Despite the simplicity of this process—hence the rapidity with which the computer could execute a single step of the algorithm—its use would not be warranted unless the priority indexing were highly meaningful. If there were strong a priori grounds for believing certain of the problem variables would be equal to one in the optimal solution, the simpler enumerative process might be applied to these variables in an initial stage of the algorithm, and for each resulting 0-1 assignment, a more complex process applied to the remaining variables.

The procedure of applying some of the tests at an earlier point in the solution sequence than the point currently visited—when a new value is found for  $b_0$  (consideration 6)†—has both advantages and disadvantages. On the positive side, if  $w_i$  is forced equal to 1 by tests applied at an antecedent branch of the tree, the term  $i$  may be inserted in the solution se-

† Balas employs a restricted version of Test 4 in this fashion in his additive algorithm.

quence directly at that point, to remain out of consideration at all later points. Thus the necessity to redetermine that  $w_i$  is forced to equal one after each of several backtracking steps would be eliminated. On the other hand, the question arises as to how far back in the solution sequence to go in applying the tests. Checking backward one step at a time from the current solution stage yields no gains since the algorithm will do that anyway. Jumping back too far—which may be only two steps back—results in a loss of time, and there is a cost of digression that may offset gains in any case. The procedure would be most likely to pay off immediately after obtaining the first value for  $b_0$ , or after obtaining a value for  $b_0$  that was appreciably smaller than the preceding value (in terms of the original problem, rather than in terms of the current problem).

#### EXTENSION OF THE ALGORITHM TO THE GENERAL INTEGER PROGRAMMING PROBLEM

THE Multiphase-Dual Algorithm may readily be extended to solve problem (1) when the  $w_i$  are integer but are not restricted to be 0 or 1. This is accomplished in the following way. We assume that for each  $w_i$  there is some known integer  $\alpha_i$  such that  $0 \leq w_i \leq \alpha_i$ . Then the solution sequence may be used for the general integer programming problem in almost exactly the same way as for the 0-1 problem. For the general problem, however, the term  $i$  no longer signifies  $w_i = 1$ , but instead denotes that  $w_i$  is assigned some integer value  $k_i$ , where  $0 \leq w_i \leq \alpha_i$ .

For simplicity, when the  $i$  term is added to the solution sequence  $w_i$  may always initially be assigned either its current upper or lower bound, and then respectively decremented or incremented at the backtracking step until it reaches its alternate bound. The tests must accordingly be modified so that instead of determining that  $w_i = 0$  or  $w_i = 1$ , they determine bounds  $p_i$  and  $q_i$  such that  $p_i \leq w_i \leq q_i$ .

The modifications of the tests of the fourth section to apply to variables with lower and upper bounds other than 0 and 1 are very nearly self-evident. However, there are various ways in which shortcuts may be effected, and certain additional considerations become relevant to the general integer programming problem that have little influence on the efficiency of an algorithm designed specifically for the 0-1 problem. Several of these considerations are presented in reference 14, where theorems are developed that show how to rule various branches of the solution tree out of consideration for the knapsack problem.

The special aspects and problems involved in binding the various considerations sketched here into a unified and specific algorithm are left to another paper, although certain preliminary computational trials indicate that a method that employs only a portion of these considerations may in

fact have practical value.† A report of these results, and the details of the extension of the Multiphase-Dual Algorithm to the general integer programming problem are to be found in reference 16.

## APPENDIX

### Proofs of Theorems

**Proof of Theorem 1.** We prove the theorem by induction, referring to the numbered steps of Diagram 1. It is obvious that the theorem is true when  $m=1$ . Under the assumption that it is true for  $m < k$ , we will show that it is true for  $m-k+1$ .

At Step 1, before any variable is assigned a value, the process goes to Step 5 and terminates only if there is no optimal solution to the problem. Thus, we assume that the process goes to Step 2 after Step 1. If the process then passes to Step 3, so that an underscored term is added to the sequence, the problem reduces to one in at most  $k$  variables. This follows from the fact that by Step 5, an underlined term will never be changed or dropped until termination unless it is preceded by a term that is not underlined. Thus, if the method must fail, it can only do so by initially visiting Steps 1, 2, 4 in that order. At Step 4 a term is selected to begin the solution sequence, and the process returns to Step 1. The problem is now temporarily reduced to a  $k$  variable problem in which the variable associated with the first term of the solution sequence is in fact no longer variable, but constant. But by Step 5, the first term of the sequence will not be changed until the succeeding terms satisfy the conditions for terminating the process when it is applied to the specified  $k$  variable problem. After this termination occurs for the  $k$  variable problem, and the first term is replaced by its underlined complement, the foregoing arguments show that termination for the entire problem coincides with termination for the new  $k$  variable problem in which the variable associated with the first term again becomes a constant by assuming the second of the two values available to it. This completes the proof, since the variable associated with the first term can only assume the two values that have been assigned to it in this process, and, by assumption, in each case the method handles the resulting  $k$  variable problem as specified by the theorem.

**Proof of the Lemma.** Parts (i) and (ii) follow immediately from the fact that for  $u$  as defined,  $\bar{w}A \geq c$  implies  $\bar{w}Au \geq cu$ . To prove (iii), we divide  $A$  into the two submatrices  $A^1$  and  $A^2$ , and divide the  $c$  vector correspondingly into  $c^1$  and  $c^2$  so that  $\bar{w}A^1 < c^1$  and  $\bar{w}A^2 \geq c^2$ . Correspondingly, we also divide  $u$  into the subvectors  $u^1$  and  $u^2$ . We observe that if  $u^1 \neq 0$ , then  $\bar{w}A^1 u^1 < c^1 u^1$ , and similarly, if  $u^2 \neq 0$ ,  $\bar{w}A^2 u^2 \geq c^2 u^2$ . Also, not both  $u^1$  and  $u^2$  can be zero. For the first part of (iii) we note that the contrary hypothesis is  $u^2 = 0$  and  $\bar{w}Au \geq cu$ . But then  $\bar{w}Au = \bar{w}A^1 u^1 < c^1 u^1 = cu$ , which is impossible. For the second part of (iii), note that if  $u^1 \neq 0$ , then

† These preliminary investigations suggest that a promising variation arises by coupling the Multiphase-Dual Algorithm with the simplex method, and enforcing parametric shifts of the objective function hyperplane, creating a composite method that combines features of both of the categories (ii) and (iv) mentioned in the introduction to this paper.

$\bar{w}A^2 u^2 = c^2 u^2$  implies  $\bar{w}Au < cu$ . Hence it follows that  $\bar{w}A_j^2 > c_j^2$  for some column  $j$  of  $A^2$ . To prove (iv), we see from (ii) that if  $w^*$  is optimal for (1), then  $w^*Au \geq cu$ , and hence if  $\bar{w}$  is optimal for (2),  $\bar{w}b \leq w^*b$ . Finally, (v) follows directly from (iv).

**Proof of Theorem 2.** The assumptions of the theorem in conjunction with part (iii) of the lemma yield  $\bar{w}A_1 - c_1 > 0$  and  $c_2 - \bar{w}A_2 > 0$ , so that  $P > 0$ . By substitution it may be verified that

$$(a) \quad \bar{w}(A_1 + PA_2) = c_1 + Pc_2.$$

If  $u_2/u_1 > P$  and  $u_2, u_1 > 0$ , then  $u_2 - Pu_1 > 0$ , and

$$(b) \quad \bar{w}(u_2 - Pu_1)A_2 < (u_2 - Pu_1)c_2.$$

Multiplying (a) through by  $u_1$  and substituting in (b) we obtain

$$\bar{w}(u_1 A_1 + u_2 A_2) < u_1 c_1 + u_2 c_2, \text{ or } \bar{w}Au < cu,$$

and hence  $\bar{w}$  is infeasible for (2) when  $u_2/u_1 > P$ . Similarly, if  $u_2/u_1 \leq P$  and  $u_2, u_1 > 0$ , it follows that  $\bar{w}Au \geq cu$ , and  $\bar{w}$  is feasible for (2).

**Proof of Theorem 3.** Let  $w^i$  denote the optimal solution to (2) obtained on the  $i$ th visit to Step 2, and let  $F_i$  and  $G_i$  denote the  $F$  and  $G$  constraints defined on the  $i$ th visit to Step 5. (Hence  $w^i$  is the optimal solution to (2) obtained relative to the  $s$ -constraint  $F_i + G_i$ , provided the process does not stop between Steps 2 and 5.) If any  $w^i$  is feasible for (1) it must by the lemma be optimal for (1), and hence the current  $s$ -constraint produces a maximum value for  $\min(wb|wa \geq c_a, w_i = 0, 1)$ . Suppose  $w^i$  is not feasible for (1).  $F_1$  is a positive multiple of the constraint of (1) that is satisfied by  $w^i$ , and  $G_1$  is a positive multiple of the constraint of (1) that is unsatisfied by  $w^i$ . By Theorem 2, using the identities  $u_1 = 1, f/g = P$ , and  $u_2 = P + \epsilon$ , it follows that  $w^i$  does not satisfy the constraint  $F_1 + (f/g + \epsilon)G_1$ , since  $u_2/u_1 = P + \epsilon > P$ . If the optimal solution  $w^2$  to (2) obtained relative to the latter constraint still satisfies  $F_1$ , then again by Theorem 2,  $w^2$  satisfies the constraint  $F_1 + G_1$ , and it follows that  $w^2 b \geq w^i b$ . Assuming that  $w^2$  is infeasible for (1), so that the process continues,  $F_2 = F_1$ . Of course,  $w^2 \neq w^i$  since  $w^i$  is infeasible for  $F_1 + (f/g + \epsilon)G_1 (= F_2 + G_2)$ . As long as Method A continues to produce  $s$ -constraints at Step 5 for which the optimal solution to (2) satisfies  $F_1$ , the  $i$ th of these constraints may be represented in the form  $F_1 + k_i G_1$ , where the scalar  $k_i$  is sufficiently larger than  $k_{i-1}$  (for  $i > 1$ ) that  $w^{i-1}$  is infeasible for the  $i$ th  $s$ -constraint. It follows as above from Theorem 2 that for all  $p < i$ : (a)  $w^p$  is infeasible for  $F_1 + k_i G_1$ , (b)  $w^i$  is feasible for  $F_1 + k_p G_1$ , (c)  $w^i b \geq w^p b$  and  $w^i \neq w^p$ . Since there are only a limited number of 0-1 solutions, eventually a solution  $w^q$  must be obtained such that either (i)  $w^q$  does not satisfy  $F_1$ , or (ii)  $w^q$  is feasible for (1). [Problem (2) must have a feasible solution by the assumption that (1) has a feasible solution.] Case (ii) produces the maximum value of  $\min(wb)$ , as already remarked. For (i), the constraint given a positive weight in defining  $G_1$  must finally be satisfied by  $w^q$ . Thus, associating the constraints satisfied and unsatisfied by  $w^i$  respectively with the satisfied and unsatisfied constraints in Theorem 2, if  $z^{q-1}$  denotes the ratio  $u_2/u_1$  defined relative to the  $q-1$ st  $s$ -constraint, and if  $z^q$  denotes  $u_2/u_1$  defined relative to the  $q$ th  $s$ -constraint, it follows that  $\bar{w}b \leq w^q b$  for any optimal solution  $\bar{w}$  to (2) obtained when  $u_2/u_1 \geq z^q$ . Similarly,  $\bar{w}b \leq w^{q-1} b$  for any optimal solution  $\bar{w}$  to (2) obtained when  $u_2/u_1 \leq z^{q-1}$ .

By selecting  $\epsilon$  small enough,  $w^*$  and  $w^{*-1}$  will be the only optimal solutions to (2) for  $u_2/u_1$  in the range  $z^\epsilon > u_2/u_1 > z^{\epsilon-1}$ . (Such a value of  $\epsilon$  exists since there are a limited number of 0-1 solutions.) With the indicated cutoff, Method A selects either the  $q-1$ st or the  $q$ th  $s$ -constraint, depending upon which yields the higher value for  $\min(wb)$ . But this  $s$ -constraint must maximize the minimum objective function value to (2) when  $\epsilon$  is chosen small enough, and the theorem is proved.

**Proof of Theorem 4.** If there exists a fractional solution  $\bar{w}$  that is feasible for (2), then the integer solution  $\bar{w}$  is also feasible for (2), where

$$\bar{w}_i = \begin{cases} \lceil \bar{w}_i \rceil & \text{if } a_i \leq 0, \\ \lfloor \bar{w}_i \rfloor & \text{if } a_i > 0. \end{cases}$$

Assume otherwise that the theorem is false. The same proof applies both to fractional and integer solutions, as follows. By assumption, if  $\bar{w}$  is an optimal solution for (2), and if there is any  $i$  such that  $b_i \leq 0$  and  $a_i \geq 0$ , or such that  $b_i \geq 0$  and  $a_i \leq 0$ , then for at least one such  $i$ , which we denote by  $p$ ,  $\bar{w}_i$  assumes a different value than specified in the theorem. Let

$$\delta = \begin{cases} 1 - \bar{w}_p & \text{if } a_p \geq 0 \text{ and } b_p \leq 0, \\ -\bar{w}_p & \text{if } a_p \leq 0 \text{ and } b_p \geq 0 \end{cases}$$

(assigning  $\delta$  either value if  $a_p = b_p = 0$ ), and let  $w^*$  be defined so that

$$w_i^* = \begin{cases} \bar{w}_i & \text{if } i \neq p, \\ \bar{w}_i + \delta & \text{if } i = p. \end{cases}$$

Then  $w_p^*$  has the value assigned to it by the theorem and

$$\begin{aligned} w^*a &= \bar{w}a + \delta a_p, \\ w^*b &= \bar{w}b + \delta b_p. \end{aligned}$$

If  $b_p \leq 0$  and  $a_p \geq 0$  then  $\delta > 0$  and  $w^*a \geq \bar{w}a$ ,  $w^*b \leq \bar{w}b$ . But this means that  $w^*$  must also be optimal for (2). Similarly, if  $b_p \geq 0$  and  $a_p \leq 0$  then  $\delta < 0$ , yielding the same conclusion. If  $w^*$  is still not compatible with the theorem we now ascribe it the role of  $\bar{w}$  above, deriving a new  $w^*$ , and repeating the process until an optimal solution is obtained whose components conform to the specifications of the theorem. This completes the proof by contradiction.

**Proof of Theorems 5A and 5B.** Since these theorems follow directly from results in references 5 and 14, we will not repeat the proofs here.

**Proof of Theorems 6A and 6B.** Theorem 6B also follows from a more general theorem by the author in reference 14, whose proof we will not bother to duplicate, and Theorem 6A follows from 6B using problem (2a).

#### REFERENCES

1. BALAS, EGON, "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," *Ops. Res.* **13**, 517-545 (1965).
2. BEN-ISRAEL, A., AND CHARNES, A., "On Some Problems of Diophantine Programming," *Cahiers du Centre L'Études de Recherche Opérationnelle*, Brussels, 1962.
3. CAMION, P., "Une méthode de résolution par l'algèbre de Boole des problèmes

- combinatoires ou interviennent des entiers," *Cahiers du Centre D'Études de Recherche Opérationnelle*, Brussels, 1960.
4. CHARNES A., AND COOPER, W. W., *Management Models and Industrial Applications of Linear Programming*, vol. II, Wiley, New York, 1961.
  5. DANTZIG, G. B., "Discrete Variable Extremum Problems," *Opns. Res.* 5, 266-277 (1957).
  6. ———, "Note on Solving Linear Programs in Integers," *Naval Res. Log. Quart.* 6, (1959).
  7. ELMAGHRABY, S. E., "An Algorithm for the Solution of the Zero-One Problem of Integer Linear Programming," Department of Industrial Administration, Yale University, May 1963.
  8. FORTET, R., "Applications de l'algèbre de Boole en recherche opérationnelle," *Revue Française de Recherche Opérationnelle* 4, 1960.
  9. GIFFLER, B., AND THOMPSON, G. L., "Algorithms for Solving Scheduling Problems," *Opns. Res.* 8, 487-503 (1960).
  10. GILMORE, P. C., AND GOMORY, R. E., "A Linear Programming Approach to the Cutting Stock Problem—Part II," *Opns. Res.* 11, 863-888 (1963).
  11. GLOVER, FRED, "A Bound Escalation Method for the Solution of Integer Linear Programs," *Cahiers du Centre D'Études de Recherche Opérationnelle* 6, Brussels, 1964.
  12. ———, "A Hybrid-Dual Integer Programming Algorithm," *Cahiers du Centre D'Études de Recherche Opérationnelle* 3, 1965.
  13. ———, "Generalized Cuts in Diophantine Programming," *O.N.R. Research Memorandum No. 132*, Graduate School of Industrial Administration, Carnegie Tech., 1964.
  14. ———, "The Knapsack Problem: Some Relations for an Improved Algorithm," *Management Sciences Research Memo No. 28*, Graduate School of Industrial Administration, Carnegie Tech., 1965.
  15. ———, "Surrogate Constraint Determination and Applications to Integer Programming," (in preparation).
  16. ———, "Extension of the Multiphase-Dual Algorithm to the General Integer Linear Programming Problem," (in preparation)
  17. ———, AND ZIONTS, STANLEY, "A Note on the Additive Algorithm of BALAS," Graduate School of Industrial Administration, Carnegie Tech, 1964; *Opns. Res.* 13, 546-549 (1965).
  18. GOMORY, RALPH L., "Solving Linear Programming Problems in Integers," OOR-AMS Symposium on Combinatorial Analysis and Designs, 1959.
  19. ———, "An Algorithm for Integer Solutions to Linear Programs," in R. L. Graves and Ph. Wolfe (eds.), *Recent Advances in Mathematical Programming*, McGraw-Hill, New York, 1963.
  20. ———, "All-Integer Integer Programming Algorithm," in J. F. Muth and G. L. Thompson (eds.), *Industrial Scheduling*, Prentice Hall, New York, 1963.
  21. ———, AND HOFFMAN, A. J., "On the Convergence of an Integer-Programming Process," *Naval Res. Log. Quart.* 10, No. 2 (1963).
  22. GORDON, B. B., HOUSE, R. W., LECHLER, A. P., NELSON, L. D., AND RADO,

- T., "Computer Studies of a Certain Class of Linear Integer Programming Problems," Battelle Memorial Research Institute, Columbus, Ohio, 1964.
23. LAND, A. H., AND DOIG, A. G., "An Automatic Method of Solving Discrete Programming Problems," *Econometrica* **28** (1960).
  24. LITTLE, J. D. C., MURTY, K. G., SWEENEY, D. W., AND KAREL, C., "An Algorithm for the Traveling Salesman Problem," *Opns. Res.* **11**, 972-989 (1963).
  25. NELSON, L. D., "On a Special Class of Problems in Integer Linear Programming," Ph.D. thesis, Department of Mathematics, The Ohio State University, 1965 (final draft in preparation).
  26. SZWARC, W., "The Mixed Integer Linear Programming Problem When the Integer Variables are Zero or One," Graduate School of Industrial Administration, Carnegie Tech, 1963.
  27. THOMPSON, G. L., "The Stopped Simplex Method: I. Basic Theory for Mixed Integer Programming; Integer Programming," *Revue Française de Recherche Opérationnelle*, 1964.
  28. ———, "The Stopped Simplex Method: II. All Solutions to Pure and Mixed Integer Programming Problems with Bounded Variables," (in preparation).
  29. YOUNG, R. D., "A Primal Integer Programming Algorithm," Ph.D. Thesis, Graduate School of Industrial Administration, Carnegie Tech, 1964.