ORC 66-29
September 1966

AN ALGORITHM FOR SOLVING THE LINEAR INTEGER PROGRAMMING PROBLEM OVER A FINITE ADDITIVE GROUP, WITH EXTENSIONS TO SOLVING GENERAL LINEAR AND CERTAIN NONLINEAR INTEGER PROBLEMS

by
Fred Glover

OPERATIONS RESEARCH CENTER

COLLEGE OF ENGINEERING

UNIVERSITY OF CALIFORNIA - BERKELEY

AN ALGORITHM FOR SOLVING THE LINEAR INTEGER PROGRAMMING PROBLEM OVER A FINITE ADDITIVE GROUP, WITH EXTENSIONS TO SOLVING GENERAL LINEAR AND CERTAIN NONLINEAR INTEGER PROGRAMS

by

Fred Glover
Operations Research Center
University of California, Berkeley

September 1966 ORC 66-29

This research was partially supported by the Office of Naval Research under Contract Nonr-222(83), The National Science Foundation under Grant GP-4593, The Army Research Office under Contract DA-31-124-ARO-D-331 and the University of California.

ABSTRACT

Ralph Gomory has recently aroused interest in a special type of knapsack problem in which the constraint coefficients and constant term are elements of a finite additive group. The significance of this problem lies in the fact that it is closely related to the general integer linear programming problem, resulting by removing the nonnegativity restrictions on those variables in the general problem that lie in an optimal basis for the associated linear program.

Gomory has shown how to solve the special knapsack problem by adapting a dynamic programming recursion originally designed for the ordinary knapsack problem, and has identified sufficient conditions under which the solution of the special knapsack problem will satisfy the nonnegativity requirements in the general integer program, thereby yielding an optimal solution to that problem as well.

In this paper we present an algorithm for solving the special knapsack problem that is capable of accommodating a variety of constraints in addition to the special knapsack constraint. Our purpose in doing this is to expand the range of problems for which the optimal solution for the special problem will also provide an optimal solution to the general integer program from which it was derived.

We develop the algorithm in a general framework that also permits a direct application to solving the general integer programming problem and certain nonlinear integer problems without attempting to solve the special knapsack problem. Some of the results developed can also be exploited by integer programming algorithms other than the one of this paper.

1. Introduction

We represent the ordinary linear programming problem as that of finding an n x 1 vector x and an m x 1 vector z to

(1) Minimize cx

subject to $Ax + z = b, \quad x \geq 0, \quad z \geq 0,$

where A, b, and c are matrices of constants with dimensions m x n, m x 1, and n x 1, respectively. We assume for convenience that the canonical form (1) has been obtained by the simplex method applied to a linear program that was initially in a corresponding form, but which in (1) also satisfies the primal and dual feasibility criteria, $b \geq 0$ and $c \geq 0$.[1] As is well known, an optimal solution to (1) is then given by $x = 0$ and $z = b$.

Hereafter, we specify the components of z and x to be integers, whereupon (1) represents a pure integer programming problem. Following Gomory [5], if we relax the requirement $z \geq 0$, the constraint set of the problem becomes $Ax \equiv b \pmod 1$, $x \geq 0$ and integer. We may replace b with any vector A_o such that $A_o \equiv b \pmod 1$, and similarly replace the jth column of A with any vector A_j which is its equivalent modulo 1, without altering the set of feasible solutions x. In particular, we may drop the integer parts of the components of A and b, so that the entries of the A_j are nonnegative and less than 1. Let us then write the relaxed integer problem as

(2) Minimize $\sum c_j x_j$

subject to $\sum A_j x_j \equiv A_o \pmod 1, \quad x \geq 0$ and integer,

and where, by assumption, $c_j \geq 0$ for all j. Given that the components

---

[1] The original problem is assumed to have a finite optimal solution.

of A^o and b^o are integers, where A^o and b^o denote the original A matrix and b vector that were transformed to obtain the primal and dual feasible form of (1), Gomory [4] has shown that the vectors u satisfying

$$u \equiv \sum_{j=0}^{n} A_j y_j \quad \text{for integer} \quad y_j$$

are elements of a finite additive group of order D, where D is the absolute value of the determinant of the matrix that was inverted by the simplex method to yield (1) in its primal and dual feasible canonical form.

2. Integer Solutions to (1) and (2)

It is clear that an optimal solution x^* to (2) will not necessarily provide an optimal solution (x^*, z^*) to (1), where $z^* = b - Ax^*$, since the components of z^* may not all be nonnegative. However, it might be hoped that the solution to (2) will extend optimally to (1) in this fashion, and Gomory [5] has indicated some of the conditions under which this will occur.

In order to solve (2), Gomory proposed a variant of a dynamic programming recursion for the knapsack problem developed jointly with Paul Gilmore [2]. Some refinements and improvements in this dynamic programming recursion applied to problem (2) have been suggested by W. W. White [6], and undoubtedly this approach merits further consideration and elaboration.

In this paper, however, we propose a different method for solving (2) that is expressly designed to accommodate a number of additional restrictions on those x_j which are admitted as feasible. The ability to handle such restrictions may be desirable either because they apply directly to

the x_j which are feasible for (1), or because they apply implicitly to the x_j in order to assure that $z \geq 0$. This ability may also be desirable if one elects to solve (1) by partitioning the less constrained solution space of (2) in a manner to yield a series of problems, the solution to one of which will give an optimal solution to (1). We develop the method of this paper in a general form that may also be directly applied to solve problem (1), and problems more general than (1). However, because the method seems particularly suited to exploiting certain structures that arise in connection with problem (2), a procedure that takes advantage of this characteristic of the method may in some instances prove more effective for solving (1) than a completely direct application of the method.

In our approach we generate a sequence S of solutions $x(i) = (x_1^i, x_2^i, \ldots, x_n^i)'$. The cost associated with x(i) is denoted by $c(i) = \sum c_j x_j^i$. Likewise, we associate with x(i) the group element $A(i) \equiv \sum A_j x_j^i$. We construct S so that it satisfies the following conditions:

1. If $p \neq q$, then $x(p) \neq x(q)$.
2. If $p < q$, then $c(p) \leq c(q)$.
3. x(i) is an optimal solution to (2) when A_o is replaced by A(i).
4. S is finite, and $A(i) \equiv A_o$ for some x(i) in S if and only if problem (2) has a feasible solution.

If we alternately interpret the A_j as ordinary column vectors, and interpret equivalence modulo 1 as ordinary equality, our strategy in

generating S may be seen to correspond quite closely to the strategy of the dual simplex method in solving the ordinary linear programming problem, since the successive basic solutions determined by the pivot rules of the dual simplex method satisfy exactly the same four conditions. Rather than employ pivot transformations to carry us from one term of the sequence to the next, however, we resort to a procedure of truncated enumeration. Truncated enumeration methods are in general characterized by an abbreviated search over a tree of all possible solutions, and derive their efficiency from the ability to exclude certain branches of the tree from consideration, so that, when the criteria for exclusion are sufficiently effective, only a small fraction of the potentially vast number of alternatives need ever be generated or examined.

The choice rule that leads to a sequence of solutions satisfying the four conditions listed above is called the least cost rule (see [3]), and is closely related to a shortest path algorithm due to G. B. Dantzig [1]. However, while the least cost rule is customarily given as finding a minimum over a generally enlarging set of nodes of a solution tree, our approach prescribes this determination over a constant or decreasing set of arc classes, the arc class j consisting of those arcs of the tree along which x_j is assigned a value.[1] The strategy underlying our algorithm is to imbed this variant of the least cost rule in a framework that permits its properties to be exploited in a computationally efficient manner.

---

[1] See [3] for a discussion of the least cost and other rules in terms of tree representation.

3. The Algorithm

We designate problem (3) to be the problem having the same objective function as (2), but with an unspecified constraint set, possibly nonlinear. Our general approach will then apply directly to solving problem (3) in nonnegative integer variables. After developing the suitable propositions underlying the method in the general case, we will subsequently introduce the specializations appropriate to solving those particular problems around which our concern centers.

We introduce the following definitions and notation. A vector x will be called a continuation of a vector y if $x = y + z$ for some integer vector $z \geq 0$. We call x a simple continuation of y if $x = y + e_j$ for some j, where e_j is the vector having its jth component equal to 1 and all remaining components 0.

In generating the sequence of solutions x(0), x(1), ... to obtain an optimal solution to problem (3), we specify that $x(0) = 0$, and that each x(i) is a simple continuation of some x(k), $k < i$. (Note that x(k+1) may not be a simple continuation, or any continuation, of x(k).) To assure that the sequence of solutions generated will satisfy this condition, and other conditions that we will subsequently impose, we associate with every nonnegative integer vector x a set of vectors R(x), and say that the vector y is a permissible continuation of x if $y = x + z$ for $z \in R(x)$. The set R(x) will be defined to consist of all nonnegative $z \neq 0$ such that $r \leq s$, where z_r is the last nonzero component of z and x_r is the first nonzero component of x ($s = n$ if $x = 0$). We also define $R_k(x)$ to be the set of all $z \in R(x)$ such that $\sum z_j = k$. Thus, the simple continuations generated from x in S are those vectors of the form $x + z$, $z \in R_1(x)$.

In what follows we will want to refer to two sequences of vectors, S and $\bar{S}$. The parent sequence S consists of every nonnegative integer vector that yields an objective function value cx less than or equal to the optimal value for problem (3). We will specify a procedure for generating the components of S in a certain order which satisfies (with slight modification) the conditions 1 - 4 of the preceding section. In addition, this order will exhibit other properties that carry over to the contracted sequence $\bar{S}$, which is actually the sequence we will be interested in generating. The procedure for generating $\bar{S}$ differs from that used to generate S in that rules are employed that permit certain subsequences of solutions in S to be bypassed without receiving consideration.

To specify the nature of these rules, we introduce the sequences S_c and $\bar{S}_c$ which denote the portions of the completed sequences S and $\bar{S}$ that are generated as of some particular stage. Thus, at the point at which the kth solution in S has just been generated, S_c consists of the x(i) for i = 0, 1, ...,k.

Just as the permissible continuations of those x in S_c are restricted by requiring that they assume the form $x + z$, $z \in R(x)$, we shall require (define) that the permissible continuations of the x in $\bar{S}_c$ assume the form $x + z$ for $z \in \bar{R}(x)$, where $\bar{R}(x)$ is some subset of R(x) to be specified subsequently.

Let S* denote the set of all optimal solutions in S, and $\bar{S}^*$ correspondingly denote the set of all optimal solutions in $\bar{S}$. Then it is evident from our discussion of S that S* must be nonempty whenever an optimal solution to problem (3) exists. We will of course want the rules for discarding continuations in $\bar{S}$ to assure that $S^* \neq \emptyset$ implies $\bar{S}^* \neq \emptyset$. As a

basis for accomplishing this we introduce two types of dominance relations.

Ordinary dominance: We will say that a vector x dominates a vector y, or xDy, if for every $z \geq 0$ such that $y + z$ is optimal for problem (3), $x + z$ is also optimal.

Local dominance: We will say that x locally dominates y with respect to the sequence $\bar{S}_c$, or $x\bar{D}y$, if $x + z$ is optimal for every $z \in \bar{R}(y)$ such that $y + z$ is optimal.

It is to be noted that local dominance depends on the particular stage in generating $\bar{S}$ to which one has reference. Generally, if at some point a vector y has no (permissible) continuations that are optimal, then y is (locally) dominated by every vector x.

We may also observe in passing that the relations D and $\bar{D}$ are both reflexive, and that they are neither symmetric nor antisymmetric. On the other hand, D is associative, but $\bar{D}$ is not. Specifically, $u\bar{D}v$, $v\bar{D}w$ implies $u\bar{D}w$ if and only if $\bar{R}^*(w) \subset \bar{R}^*(v)$, where, for any x, $\bar{R}^*(x)$ is the (possibly empty) subset of $\bar{R}(x)$ such that $z \in \bar{R}^*(x)$ implies $x + z$ is optimal.

There are two principal rules based on the foregoing remarks that will enable the procedure for generating S to be transformed into the procedure for generating $\bar{S}$. They are:

Rule 1. If $x \in \bar{S}_c$, $y \notin \bar{S}_c$, and xDy, then all v such that $v \geq y$ may be eliminated from $\bar{S}$.

Rule 2. If $x \in \bar{S}_c$, $y \notin \bar{S}_c$, and $x\bar{D}y$, then y and all v such that $v = y + z$, $z \in \bar{R}(x)$, may be eliminated from $\bar{S}$.

By reference to Rules 1 and 2, we now define $\bar{R}(x)$ (or, conveniently, $\bar{R}(u)$). Specifically, for $u \in S$, define $\bar{R}(u)$ relative to $\bar{S}_c$ to be that

subset of $R(u)$ such that, $w \in (R(u) - \bar{R}(u))$ if and only if there exist $x \in \bar{S}_c$, $y \notin \bar{S}_c$, where y is an earlier component of S than u, such that, for $v = u + w$,

(i) xDy and $v \geq y$, or

(ii) $x\bar{D}y$ and either $v = y$ or $v = y + z$, $z \in \bar{R}(y)$.

It may be noted that the requirement that y is an earlier component of S than u avoids certain problems of circularity, and, as we will see, also brings the definition of $\bar{R}(u)$ into correspondence with the information available from the algorithm at the point at which a decision is made to include or exclude u from $\bar{S}$. That is, $\bar{R}(u)$ limits the permissible continuations of u, at the time u is under consideration as a potential element of $\bar{S}$, to those permissible continuations of u in S that do not qualify to be dropped by Rule 1 or Rule 2.

The procedure for generating S and $\bar{S}$ is then as follows. Associated with the vector $x(i) = (x_1^i, x_2^i, \ldots, x_n^i) \in S$, we define $d_i = \text{Min}(j: x_j^i \geq 1)$ if $i \geq 1$, and $d_o = n$. Thus, $z \in R_1(x(i))$ if and only if $z = e_j$ for some $j \leq d_i$.

We also wish to identify the vector $x(t_j)$ such that the next vector $x(k)$ of the form $x(i) + e_j$ to be added to S_c will be given by $x(k) = x(t_j) + e_j$ for some j. The identification is accomplished (as will be proved) if we initially define $t_j = 0$ for all j, and then define the "next value" $\bar{t}_j$ of t_j so that $\bar{t}_j = \text{Min}(i: i > t_j \text{ and } j \leq d_i)$, where t_j is assigned its next value $\bar{t}_j$ immediately after the solution $x(t_j) + e_j$ is added to S_c. It is possible that some of the t_j may not be well-defined at a particular stage of generating S, and consequently we must also identify the set T relative to a given S_c which consists of those j such that t_j is currently well-defined.

Finally, to determine the j which is to contribute the next vector in S_c, we define the "next cost" associated with each j to be $N_j = c(t_j) + c_j$. The foregoing definitions of d_i, t_j, T and N_j are understood to apply to $\bar{S}$ as well as to S by replacing each occurrence of S and S_c respectively with $\bar{S}$ and $\bar{S}_c$.

The Algorithm

Beginning with $x(0) = 0$, assume that $x(i)$ has been generated for $i = 0, 1, \ldots, k-1$. The procedure to follow is designed to generate S with instruction 2 removed, and to generate $\bar{S}$ with instruction 2 present.

1. If T is empty, (3) has no feasible solution and there is no kth term to be generated. Otherwise, let J be determined by $N_J = \text{Min}(N_j: j \in T)$.[1] If there is more than one candidate for J, select the one for which J is the smallest.
2. If the solution $x(t_J) + e_J$ satisfies the criteria to be eliminated by Rule 1 or Rule 2, assign t_J its next value $\bar{t}_J$ and return to 1. Otherwise,
3. Let $x(k) = x(t_J) + e_J$. If $x(k)$ is feasible for (3) (hence the first feasible solution found), then $x(k)$ is optimal and the problem is solved. Otherwise,
4. Assign t_J its next value $\bar{t}_J$, increment k by 1, and return to 1.

There are several things to be observed about the general framework of the method as described above. First, the process specified in instruction 2, whereby $x(t_J) + e_J$ is bypassed, corresponds precisely to setting $\bar{R}(x(t_J) + e_J) = \emptyset$, since failing to include a solution in $\bar{S}$ immediately eliminates all permissible continuations (though not all continuations) of

---

[1] It is not necessary to compute a minimum over all elements of T each time N_J is determined, since only one of the N_j is changed at each step.

that solution. When Rule 1 applies to justify the elimination of all continuations of a solution, the method thus accomplishes this by successively eliminating certain sets of permissible continuations.

Second, the description of the method does not specify either the record keeping or the tests underlying Rules 1 and 2 that enable the elimination of solutions. We reserve this specification to later sections, where we introduce several important modifications and refinements of the general framework outlined above.

It may also be observed that, while we have earlier defined $d_i = \text{Min}(j: x_j^i \geq 1)$, each d_i may be determined more simply by assigning d_k the value J at instruction 4. This is clearly true for d_1, and in fact for any $d_k = J$ such that $t_J = 0$. Assuming that it is true for all d_i, $i \leq k-1$, it must also be true for $i = k$ since, if $t_J \neq 0$, then $t_J = p$ for some $p < k$ such that $d_p = J$. But then $J = \text{Min}(j: x_j^p \geq 1)$ and $x(k) = x(p) + e_J$, from which $d_k = J$ follows.

4. Justification of the Algorithm: Propositions and Proofs

We have informally made a number of claims about the sequences S and $\bar{S}$, and have attempted to introduce considerations to make some of these claims intuitively plausible. We will now present our claims more formally, explicitly defining S and $\bar{S}$ to be those respective sequences generated by the above method with instruction 2 removed and with instruction 2 present.

We will require in what follows tnat problem (3) is bounded and that $c > 0$. Given rational $c_j \geq 0$, this latter can be assured, for example, by replacing c by a sufficient positive multiple of itself to assure all components are integer, and then replacing those $c_j = 0$ by $1/P$, where P is a number such that $\sum_{c_j=0} x_j \leq P - 1$. Thus, when solving problem (2)

it suffices to multiply c by D, and also to let P = D, where D is the value of the determinant referred to in Section 1.

Our first two propositions are essentially the same as the statements of conditions 1 and 2 in Section 2.

Proposition 1. If $x(p), x(q) \in S$, or if $x(p), x(q) \in \bar{S}$, $p \neq q$, then $x(p) \neq x(q)$.

Proof. The same proof applies both to S and $\bar{S}$. Suppose that the proposition is false, and let q assume the least value such that $x(q) = x(p)$ for some $p < q$. Since $q > 0$, $x(q) = x(h) + e_r$ for some $h < q$ and $r = d_q$. Likewise, since $x(p) = x(q) \neq 0$, we have $p > 0$ and $x(p) = x(i) + e_s$ for some $i < p$ and $s = d_p$. But by the definition of d_p and d_q it follows that $r = s$ and hence $x(i) = x(h)$. We must have $i = h$, for otherwise a duplication occurred before x(q) was generated, contrary to assumption. But $i = h$ is also impossible, for, given that $t_s = i$ when x(p) was generated, t_s was then strictly increased when assigned its next value $\bar{t}_s$, and never subsequently decreased, so that the value h of t_s (also t_r) when x(q) was generated could only have been larger than i.

Proposition 2. In either S or $\bar{S}$, if $p < q$ then $c(p) \leq c(q)$.

Proof. Again, the same proof applies both to S and $\bar{S}$. Note to begin with that $c(0) \leq c(1)$ and that the minimum value of the N_j after generating x(1) is at least as large as before. We assume that these two conditions hold for all $x(i)$, $i \leq k - 1$, and prove that they must also hold when generating x(k). For some h, we have $c(k-1) = N_h = c(t_h) + c_h$. If $\bar{t}_h \leq k - 1$, we have the next value of N_h given by $c(\bar{t}_h) + c_h \geq N_h$ since $t_h < \bar{t}_h \leq k - 1$. Thus the values for the N_j are the same or, in the case of N_h, possibly increased at the step

which generates x(k), hence the minimum value of the N_j is also nondecreasing and $c(k) \leq c(k-1)$. To complete the proof we must consider the case in which $\bar{t}_h > k-1$. In this instance $\bar{t}_h$ is not defined upon generating x(k-1) and hence the set T is decreased by one element when generating x(k). The minimum available N_j is consequently nondecreasing and again (if any N_j remain) $c(k) \leq c(k-1)$. In this fashion the c(i) continue to be nondecreasing until the undefined $\bar{t}_h$ (or some other undefined $\bar{t}_h$) becomes defined upon generating, say, x(q). But then $N_h = c(q) + c_h$, which is at least as large as the value of N_j that produced c(q) (actually exceeding this value for $c_h > 0$), so that the minimum available N_j remains nondecreasing at all steps.

While the proofs of the two preceding propositions did not depend upon $c > 0$ or the precise tiebreaking rule specified by the method for the choice of J, the proof of the next proposition requires both of these.

Proposition 3. In S and $\bar{S}$, $c(p) = c(q)$ and $p < q$ implies x(p) is lexicographically larger than x(q).

Proof. By x(p) lexicographically larger than x(q), for $x(p) \neq x(q)$, we mean that the first nonzero component of x(p) - x(q) is positive. Assume the proposition false, and denote d_p by r and d_q by s. We may assume $s < r$, for if $s = r$ the same assumptions concerning the relation between x(q) and x(p) must also apply to $x(\bar{q})$ and $x(\bar{p})$ (and conversely), where $x(\bar{q}) = x(q) - e_s$ and $x(\bar{p}) = x(p) - e_r$. Hence, replacing q and p by $\bar{q}$ and $\bar{p}$ a finite number of times permits the assumption $s < r$. But then, N_s was not defined when x(p) was created, or else, by the tiebreaking rule for the choice of J, x(q) would have

been generated before x(p). Consequently, when N_s becomes defined, $c(t_s) \geq c(p)$, and hence $c(q) \geq c(p) + c_s > c(p)$, contrary to assumption.

The next proposition is the first that does not apply to both S and $\bar{S}$.

Proposition 4. If $x(i) \in S$, then $\bar{x} \in S$ for every nonnegative integer $\bar{x}$ such that $c\bar{x} < c(i)$.

Proof. Assume on the contrary that there exists a nonnegative integer $\bar{x}$ such that $c\bar{x} < c(i)$ and $\bar{x} \neq x(p)$ for all $p < i$. We shall restrict attention to those x(p) such that, for some $r \leq n$, $x_j^p = \bar{x}_j$ for all $j > r$, $x_r^p < \bar{x}_r$, and $x_j^p = 0$ for $j < r$. From among the solutions x(p) so restricted, we identify the solution x(q) which is "closest" to $\bar{x}$ as follows. We stipulate that x(q) is that solution for which r is the smallest and such that, from among those x(p) with the same value of r, $x_r^q = \max(x_r^p)$. Now $d_q \geq r$, and hence when x(q) is generated $t_r \leq q$, for either $t_r < q$ or t_r was undefined at the point at which x(q) was generated, in which case $t_r = q$. Moreover, $t_r \leq q$ implies $N_r \leq c(q) + c_r \leq c\bar{x}$. Thus, since $d_q \geq r$, and since instruction 2 of the algorithm is removed, before generating any x(i) such that $c(i) > c\bar{x}$, the method must first generate from t_r and x(q) the solution $x(h) = x(q) + e_r$. But then x(h) qualifies as one of the restricted solutions x(p), and is "closer" to $\bar{x}$ than x(q), contrary to the choice of x(q).

Remark 1. If problem (3) has a finite optimal solution, then $S^* \neq \emptyset$.

Proof. When generating S, it follows from Propositions 3 and 4 that T can be empty only if problem (3) has no feasible solution. (In fact, with instruction 2 removed t_1 must always be well defined and T cannot

be empty in any event.) Suppose that the optimal objective function value is c^*, but that no optimal solution is obtained when generating S. Then since $c > 0$, Propositions 1 and 2 imply that $c(i) > c^*$ for some $x(i) \in S$. But then, by Proposition 3, each $\bar{x}$ such that $c\bar{x} = c^*$, $\bar{x} \geq 0$ and integer, will have been generated as $x(p)$ for some $p < i$, contrary to assumption.

Having derived the foregoing properties of the sequences S and $\bar{S}$, and having verified that $S^* \neq \emptyset$ when problem (3) has an optimal solution, we now wish to establish that $\bar{S}^* \neq \emptyset$ as well. To this end we state

Proposition 5. Assume that an external cutoff rule is applied to the generation of $\bar{S}$, so that, if the method does not stop sooner it will stop after generating M terms for M arbitrarily large but finite. Also assume that the generation of S is allowed to continue until a solution is generated with an objective function value larger than that associated with every solution in $\bar{S}$. Then for S and $\bar{S}$ so determined, $\bar{S}$ is a subsequence of S.

Proof. It is evident from Propositions 1 and 2 that S and $\bar{S}$ are well-defined by the assumptions of the proposition. Denote those $x(i)$ in S by $x^1(i)$ and those $x(i)$ in $\bar{S}$ by $x^2(i)$. Let $\hat{S}$ be the subsequence of S obtained by deleting from S each $x^1(i)$ such that $x^1(i) \neq x^2(k)$ for all $x^2(k) \in \bar{S}$. We note by Propositions 2 and 4 that the components of $\hat{S}$ must correspond to those of $\bar{S}$, though perhaps in a different order. Thus, designate the smallest i such that $x^1(i) \in \hat{S}$ by $\bar{0}$, the next smallest i by $\bar{1}$, and so on. Then we wish to prove that $x^1(\bar{i}) = x^2(i)$ for all $x^2(i) \in \bar{S}$. Suppose otherwise, and let $p = \text{Min}(i: x^1(\bar{i}) \neq x^2(i))$. Also identify the indices q and r such that $x^1(\bar{p}) = x^2(q)$ and $x^2(p) = x^1(\bar{r})$. It is assured by Proposition 1 that $q, r > p$.

Since $\bar{i} = \bar{s}$ if and only if $i = s$, for all i and s, we have $x^1(\bar{p}) = x^1(\bar{h}) + e_u$ for some u and some $h < p$, and $x^2(p) = x^2(k) + e_v$ for some v and some $k < p$. Now, $x^1(\bar{p})$ was generated before $x^1(\bar{r})$, but $x^1(\bar{r}) = x^2(p)$ implies $x^1(\bar{r}) = x^1(\bar{k}) + e_v$. Since $k < p$, this means that when $x^1(\bar{p})$ was generated, t_v ($= \bar{k}$) was well-defined. Thus there was a choice to make between generating $x^1(\bar{p})$ and $x^1(\bar{r})$. Similarly, $x^2(p)$ was generated before $x^2(q)$, but $x^2(q) = x^2(h) + e_u$, so that, by analogous reasoning, there was a choice to make between generating $x^2(p)$ and $x^2(q)$ when $x^2(p)$ was generated in $\bar{S}$. But $\bar{p} < \bar{r}$ thus implies $q < p$, providing a contradiction.

To demonstrate that $S^* \neq \emptyset$ implies $\bar{S}^* \neq \emptyset$, we prove the somewhat stronger result to follow.

Proposition 6. If $z \geq 0$ and $y + z$ is optimal for some $y \geq 0$, then there exists a solution u in $\bar{S}$ such that $u + z$ is optimal.

To prove Proposition 6 we must for the first time take into account the properties of the relations D and $\bar{D}$, and hence of the sets $R(x)$ and $\bar{R}(x)$.

We begin by observing three things, the proofs of which are immediate from our earlier definitions.

Remark 2. If $y \in R_h(x)$, $z \in R_k(x)$, then $y + z \in R_{h+k}(x)$.

Remark 3. If $y \in R_h(x)$, $z \in R_k(y)$, then $y + z \in R_{h+k}(x)$.

Remark 4. $R(y + z) \subset R(y), R(z)$.

It may be noted that noted that Remarks 2 and 3 are also true with the subscripts h and k removed. We then have

Remark 5. If, relative to some $\bar{S}_c$, Rule 2 prescribes the elimination of all solutions of the form $u + w$, $w \in \bar{R}(u)$, then the elimination of

all solutions of the form u + w, w ∈ R(u) is prescribed by Rule 1 or 2.

The validity of this remark follows directly from the definition of $\bar{R}(u)$. We may observe also that when a solution u is dropped at instruction 2 of the algorithm, it follows from Propositions 2 and 5 that no solutions of the form u + w, w ∈ R(u), will be generated in $\bar{S}$. Our next two remarks, while less immediate than those preceding, are fundamental to the proof of Proposition 6.

Remark 6. Relative to a given $\bar{S}_c$, let V be the set of all v ∈ S such that there exist x ∈ $\bar{S}_c$, y ∉ $\bar{S}_c$, for which v, x, and y satisfy condition (i) or (ii) in the definition of $\bar{R}(u)$. Then, if u = v + w for some v ∈ V and some w ∈ R(v), it follows that u ∈ V. (Here u and w do not necessarily correspond to the u and w in the definition of $\bar{R}(u)$.)

Proof. If v satisfies condition (i), then it is immediate that u does also and hence u ∈ V. If v satisfies condition (ii) for v = y, then u = y + w, w ∈ R(y), and it follows from Remark 5 that u ∈ V. Thus, we examine the case where v satisfies condition (ii) for v = y + z, z ∈ $\bar{R}(y)$. Then from the definition of u we have u = y + z + w, w ∈ R(v). Making use of the properties of R(u), by Remark 4 we have R(v) = R(y + z) ⊂ R(y), hence w ∈ R(y). But also z ∈ R(y), and by Remark 2, w + z ∈ R(y). Finally, by Remark 5, u ∈ V.

Remark 7. For V defined as in Remark 6, S - $\bar{S}$ = V.

Proof: It is obvious that V ⊂ S - $\bar{S}$. To prove the converse, suppose u ∈ S - $\bar{S}$ but u ∉ V. Moreover, let u = x(h) be the first solution in S satisfying these two conditions. Identify the solution x(i) ∈ S,

$i < h$, such that $x(i) + e_r = x(h)$ and $r = d_h$. We observe by Remark 6 (for $x(i) = v$ and $e_r = w$) that $x(i) \notin V$. However, since $x(h)$ is the first solution in S that is not in V and also not in $\overline{S}$, it follows that $x(i) \in \overline{S}$. (For convenience we assume that the indices of the solutions in $\overline{S}$ are changed, if necessary so that $x(i) \in \overline{S}$ denotes the same solution as $x(i) \in S$. Proposition 5 permits this assumption to be made without danger.) Since $r \leq d_i$, at some point in generating $\overline{S}$ we have $t_r = i$ and $N_r = c(h) = c(t_r) + e_r$. Also, since $x(h) \notin V$, $x(h)$ does not satisfy the criteria to be dropped at instruction 2. On the other hand, if the algorithm generating $\overline{S}$ halts before $x(h)$ is added to $\overline{S}$, then it cannot be due to $T = \emptyset$, since t_r must remain well-defined from the point at which $x(i)$ is generated until after $x(h)$ is produced. But the method also cannot halt in consequence of finding an optimal solution before generating $x(h)$ since, otherwise, by Proposition 5, such an optimal solution would have preceded $x(h)$ in S, and thus S would also have ended before generating $x(h)$. There are no other ways that $x(h)$ could be prevented from belonging to $\overline{S}$, and the proof is completed by contradiction.

With the foregoing remarks we have established all but one portion of the proof of Proposition 6, which we now complete as follows.

Proof of Proposition 6. We claim that for any $z \geq 0$, the lexicographically largest u such that $u + z$ is optimal is always retained in $\overline{S}$. For, suppose it is not. We may first observe that $u \in S$. This is clearly true for $z \neq 0$ by Propositions 2 and 4, and is also true for $z = 0$ since by Proposition 3, the first optimal solution in S

(hence the only optimal solution in S by the rule for termination in instruction 3) is that which is lexicographically largest. Given that $u \in S$, if $u \notin \bar{S}$ then $u \in V$ by Remark 7. Thus there exist x and y as in Remark 6 such that: (i) xDy and $u = y + w$, $w \geq 0$; or (ii) $x\bar{D}y$ and $u = y + w$ for $w \in \bar{R}(y)$ or $w = 0$. In both cases, $y + w + z$ is optimal, and so is $x + w + z$. Thus $cy = cx$, and by Proposition 3 x is lexicographically larger than y. But then $x + w$ is legicographically larger than $y + w = u$, contrary to the definition of u.

It is clear that Proposition 6 implies $\bar{S}^* \neq \emptyset$ when problem (3) has a finite optimal solution. In order to gain a clearer understanding of our foregoing results it is perhaps worth pausing to consider not only what has been proved but what has not been proved. For example, if $c \geq 0$ but $c \not> 0$, it is clear that S might not be finite, although a slight modification in generating S that allowed upper bounds to be taken into consideration would assure finiteness for an explicitly bounded problem. On the other hand, unless $c > 0$ it is easy to construct examples for which Propositions 3 and 6 are false. This does not immediately imply that $\bar{S}^* = \emptyset$, but it may actually make the generation of $\bar{S}$ less efficient by allowing a solution x that is lexicographically smaller than a solution y to be retained in $\bar{S}$ while causing y to be dropped; whereas, had y been generated first (and $y\bar{D}x$), it would have caused x to drop. But the fact that x is lexicographically smaller than y means that it will in general have a larger number of permissible continuations, forcing the method to examine more solutions than might otherwise be the case.

A more subtle point concerns the rule in instruction 1 for breaking ties in the determination of J, i.e., by taking J at its smallest possible

value.[1] It is tempting to believe that any rule for breaking ties might do as well, but this is false. (There is, of course, the class of permissible tiebreaking rules that may be made the same as the one specified by a suitable reindexing of the variables.) The choice of J not only affects the efficiency in the generation of $\bar{S}$ (in the manner just indicated for $x > 0$), but is crucial in determining whether an optimal--or even a feasible--solution will be found. It is in fact possible to specify choice rules for certain problems that will cause $\bar{S}^*$ to be empty even though an optimal solution exists.

Having established the validity of the general framework of the algorithm given above, we will subsequently present a principal variation of this method in which it is possible to obtain an optimal solution by generating only a subset of $\bar{S}$. However, before introducing this variation, and the additional propositions upon which it is based, we will now examine how the foregoing method may be applied in certain specific situations.

5. Applying the Method to Problem (2) With and Without Additional Constraints

We first consider solving problem (2) in the absence of additional constraints. Denoting the solution $x(t_j) + e_j$ of instruction 2 by $\bar{x}$, it is clear that $x(i)D\bar{x}$ for $x(i) \in \bar{S}_c$ (i.e., for $i \leqq k - 1$) if $\sum A_j \bar{x}_j \equiv \sum A_j x^i_j$ (or $A(t_J) + A_J \equiv A(i)$), since it is assured by

---

[1] The consequence of this choice may be interpreted as a secondary cost perturbation that increments each c_j by $c_j/P - \epsilon_j$, where, given that all feasible x satisfy $\sum x_j < D$, P is selected so that $P \geqq D(kc_j)$ for all j, where k is a multiple such that kc_j is an integer for all j, and each ϵ_j satisfies $0 < \epsilon_j < c_j + c_j/P$ and $\epsilon_{j+1} \leqq \epsilon_j/D$. In conjunction with the other rules of the method, this assures that $c(p) < c(q)$ for $p < q$.

Propositions 2 and 4 that $c(i) \leq c\bar{x}$. Moreover, $\bar{x}$ then qualifies to be dropped by Rule 1, since Propositions 1 and 4 imply $\bar{x} \notin \bar{S}_c$. Thus, to determine whether $\bar{x}$ may be dropped at instruction (2) it suffices in this case to check only whether $A(t_J) + A_J \equiv A(i)$.

However, dropping $\bar{x}$ in this fashion eliminates only its permissible continuations, whereas, according to Rule 1 (and Proposition 6) it is legitimate to drop all the continuations of $\bar{x}$. We will now consider a way that will permit this to be accomplished conveniently when $\bar{x}$ has the form ke_s.

Denote the value d_i associated with a solution $x(i)$ by r, so that x_r^i denotes the first nonzero component of $x(i)$. Note first of all that it is unnecessary to record any solution vector $x(i)$ since $x(i)$ may always be reconstructed by knowing the group element $A(i)$ and the value of r ($= d_i$) for each i. Thus to determine $x = x(i)$, we begin with $x = 0$, and define the next value of x to be $\bar{x} = x + e_r$. Thereupon, one finds p, $p < i$, such that $A(p) \equiv A(i) - A_r$, and repeats the process, treating $\bar{x}$ as x and $A(p)$ as $A(i)$. As soon as $A(0)$ is reached, $x(i)$ is completely determined.

This process may be slightly speeded if x_r^i is known for each i. Keeping track of this value, which we denote by U_i, is particularly easy, since, in generating $x(i) = x(p) + e_r$ we have $U_i = U_p + 1$ if $d_p = r$ and $U_i = 1$ otherwise. However, we will also wish to know the value U_i of x_r^i in order to exploit Rule 1 more fully, in accordance with our previous discussion.

Thus, given that $\bar{x}$ is a solution that qualifies to be dropped from $\bar{S}_c$ by Rule 1, where exactly one component of $\bar{x}$, say $\bar{x}_s$, is positive, then $x_s \leq \bar{x}_s - 1$ may be required of all solutions generated. This restriction,

which has the form $x_j \leq B_j$, can be accommodated in a straightforward way by modifying the definition of the next value $\bar{t}_j$ of t_j. Specifically, let $\bar{t}_j = \min(i: i > t_j \text{ and } \bar{d}_i \geq j)$, where $\bar{d}_i = r$ $(r = d_i)$ if $U_i = B_r - 1$, and $\bar{d}_i = r - 1$ if $U_i = B_r$. (One may record $\bar{d}_i$ at the same time as recording d_i, or simply flag d_i to indicate whether $\bar{d}_i = d_i$ or $\bar{d}_i = d_i - 1$.) This modified definition of $\bar{t}_j$ assures that one will not generate any solution $x(p)$, $x(p) = x(i) + e_j$, such that $x_j^p > B_j$. For the value of $j \leq \bar{d}_i \leq d_i$ implies either $x_j^i = 0$ or $j = d_i = r$ and $x_j^i \leq B_j - 1$. In either case, $x_j^p = x_j^i + 1 \leq B_j$. (Variables with 0 upper bounds are assumed dropped from the problem.) Also, the method will not fail to generate any solution in which $x_j^p \leq B_j$, provided this solution would have been generated under the original definition of $\bar{t}_j$, since the new and old definitions correspond for all $j \leq r$ whenever $x_j^i \leq B_j - 1$ $(x_j^i = x_j^p - 1)$.

In direct extension of our foregoing remarks, we next consider problem (2) augmented by constraints of the form $x_j \leq B_j$, where the B_j do not all arise naturally--i.e., as a consequence of Rule 1--but rather occur as exogenous restrictions which may apply, for example, to the x_j that solve problem (1). In this event, it may no longer be true that $A(i) \equiv A(t_j) + A_j$ implies $x(i)D\bar{x}$ $(\bar{x} = x(t_j) + e_j)$. However, suppose that, if instruction 2 were temporarily bypassed, $\bar{x}$ would be generated as $x(q)$, and that there exists some $x(p) \in \bar{S}_c$, $p < q$, such that $A(p) \equiv A(q)$. Then if either $d_q < d_p$, or if $d_q = d_p$ and $U_q \geq U_p$, it follows that $x(p)\bar{D}x(q)$, and thus $\bar{x}$ can be dropped at instruction 2 as a consequence of Rule 2. Also, it is evident that $x(p)D\bar{x}$, and $\bar{x}$ can be eliminated by Rule 1, if for any optimal solution of the form $x = \bar{x} + z \leq B$ ($z \geq 0$ and B the vector of upper bounds), it follows that $x(p) + z \leq B$, since then $x(p)$ can legitimately replace $\bar{x}$ in such a solution. We mention three things that will

assure this, in order of increasing restrictiveness, but also of increasing difficulty of application: (i) $x_j^p = 0$ for those j such that B_j is exogenous, (ii) $x_j^p \leq \bar{x}_j$ for such variables, (iii) $c\bar{x} + \min(c_j(B_j - x_j^p + 1))$ exceeds an upper bound on cx, where the minimum is computed over those j such that B_j is exogenous and $x_j^p > \bar{x}_j$.[1] This latter criterion may also be applied replacing c by e (the vector of ones) if an upper bound on $\sum x_j$ is known.

If none of the foregoing criteria apply (other than $A(p) \equiv A(q)$), it is possible that $\bar{x}$ should not be dropped, but retained as x(q) in $\bar{S}$. In this event one may, if desired, cut down the number of solutions examined by not allowing the consideration of any solution $x(q) + e_j$ such that $j < d_p$.[2] An additional index, m_q, assigned to x(q) (generally, m_i assigned to x(i)) will accomplish this if one defines $\bar{t}_j$ so that $\bar{t}_j = \min(i: i > t_j, \ m_i \leq j \leq \bar{d}_i)$, where, in this case, $m_q = d_p - 1$. The introduction of m_i and $\bar{d}_i$ in the revised definition of $\bar{t}_j$ has precisely the effect of restricting the set $\bar{R}(x(i))$ from which the permissible continuations of x(i) with respect to $\bar{S}_c$ are determined.

The foregoing remarks, however, do not take advantage of the fact that the A(i) are elements of a finite group. This is an important consideration, and we will show how to exploit its implications later. For the moment, however, we continue to evolve the structure of the algorithm

---

[1] To prove this suppose $u = \bar{x} + z$ is optimal for some $z \geq 0$. Then $cu \geq c\bar{x} + c_j z_j$ for all j. Now, x(p) + z will be optimal unless, for some j such that B_j is exogenous, $z_j + x_j^p \geq B_j + 1$. But then $x_j^p > \bar{x}_j$, $z_j \geq B_j - x_j^p + 1$, and hence $cu \geq c\bar{x} + \min(c_j(B_j - x_j^p + 1))$. When this cannot be satisfied, as stipulated in (iii), then $x(p)D\bar{x}$ is assured.

[2] Some or all of those solutions for $j = d_p$ may also be eliminated, the details of which depend on the value of U_p, ^{p} and may be readily developed from the considerations discussed here.

to accommodate other kinds of restrictions.

It is rather transparent that a restriction of the form $\sum x_j \leq M$ may be handled simply by recording $M_i = \sum x_j^i$ -- i.e., $M_i = M_p + 1$ where $x(i) = x(p) + c_j$ for some j. Then when $M_i = M$, it is desired that $\bar{R}(x(i)) = \emptyset$, which may readily be assured by setting $\bar{d}_i = 0$.

One criterion for dropping solutions at instruction 2 is immediate. If $x(p)D\bar{x}$ or $x(p)\bar{D}\bar{x}$ in the absence of the constraint $\sum x_j \leq M$, then $M_p \leq \sum \bar{x}_j$ ($= M_{\bar{x}_J} + 1$) assures that the same relation will hold in the present instance. On the other hand, if $M_p > \sum \bar{x}_j$, then $\bar{x}$ may have to be retained as $x(q)$. An exception occurs when $c\bar{x} + (M + 1 - M_p)(\text{Min}(c_j))$ exceeds an upper bound on cx, by reasoning similar to that of footnote 1 on the preceding page. It may be noted that this criterion may be checked quite easily. The criterion may also be sharpened, though at some computational expense, by indexing the c_j in ascending order of magnitude and defining k to be the least index such that $\sum_{j \leq k} B_j > M + 1 - M_p$. One then tests whether $\bar{M}$, given by $\bar{M} = c\bar{x} + \sum_{j<k} B_j c_j + (M+1 - M_p - \sum_{j<k} B_j)c_k$, exceeds the bound on cx. Moreover, if criterion (iii) on page 22 holds, then $z_j \leq B_j - x_j^p$ (see the associated footnote), and one may replace B_j in $\bar{M}$ by $\bar{B}_j = B_j - x_j^p$, redefining k in terms of the $\bar{B}_j$ instead of the B_j. We give attention to such criteria because, in the form of the algorithm introduced in the next section, upper bounds on cx are likely to be obtained before determining an optimal solution.

Now consider a restriction of a somewhat different sort. Suppose that the problem variables are divided into a number of different sets Q_h, such that at most one variable in any particular set may assume a positive value.

A familiar example of such a restriction is given by a set of constraints of the form $\sum_{j \in Q_h} x_j \leq 1$.

One method of accommodating such a restriction is as follows. Assign a different prime number to each set Q_h, and assign to the variable x_j the set of prime numbers corresponding to those Q_h to which j belongs. If the Q_h are disjoint, x_j will thus be assigned at most one prime number. Then, to each A(i) generated by the method we attach a number P_i. We let $P_o = 1$, and determine P_q, where $x(q) = x(p) + e_j$, so that $P_q = P_p G_j$, where G_j is the product of those primes associated with x_j. We do not allow x(q) to be created, however, if any prime associated with x_j divides P_p. It is easy to see that this approach will cause the method to generate only those solutions satisfying the indicated restrictions. Also, the largest P_i will not exceed the product of the primes attached to the Q_h, and may be substantially less than this product.[1]

Another (perhaps better) way of handling such restrictions is to assign the numbers 1, 2, 4, 10, 20, 40, 100, 200, 400, etc., one-to-one to the sets Q_h, and to assign each x_j, as before the numbers attached to those Q_h for which $j \in Q_h$. In this case we let $P_o = 0$, and let $P_q = P_p + G_j$, where $x(q) = x(p) + e_j$ and G_j is the sum of the numbers associated with x_j. It is not hard to see how to determine whether it is permissible to generate the solution $x(p) + e_j$ when using this scheme. Also, a quite

---

[1] The manner of assigning primes to the Q_h will significantly influence the size of the largest P_i. A step in the right direction would be, for example, to assign the smallest primes to those Q_h with the largest number of elements. It would also be possible to use -1 for the "first prime," followed by 2, 3, etc.

large number of sets Q_h may be accommodated by representing a large P_i by more than a single number, e.g., by P_i^o, P_i^1, P_i^2 where P_i is the "augmented number" (P_i^2, P_i^1, P_i^o).

The ability to drop solutions at instruction 2 is rather limited when accommodating restrictions such as the one above unless one is willing to apply more involved tests for determining $x(p)D\bar{x}$ and $x(p)\bar{D}\bar{x}$. However, if the x_j are not subject to other types of restrictions, then one criterion that assures $x(p)Dx(q)$ (representing $\bar{x}$ as $x(q)$) is $A(p) \equiv A(q)$ whenever P_p divides P_q and the P_i are determined by the prime number assignment.

6. A Variation of the Algorithm

We now indicate a refinement of the method that in some cases will make it possible to substantially reduce the number of solutions generated. First, suppose that we wish to solve problem (2) in the absence of additional restrictions, and assume that a list $g_1, g_2, \ldots, g_D$ is associated with the D elements (including 0) of the additive group, where $g_k = i$ if $A(i)$ is the kth group element and $g_k = 0$ if the kth group element has not yet been generated. Such a list would evidently be useful in the ordinary procedure of determining when a solution is to be dropped at instruction 2. However, note that if $A(p)$ and $A(q)$ are generated such that $A(p) + A(q) \equiv A_o$, then $x(p) + x(q)$ is a feasible solution to problem (2). Moreover, if x^* is an optimal solution to (2) such that either $x^* - x(p) \geq 0$ or $x^* - x(q) \geq 0$, then it can readily be shown that $x(p) + x(q)$ is an optimal solution to (2). Consequently, $A(p) + A(q) \equiv A_o$ allows us to limit the $A(i)$ generated in two ways. First, we need not generate any continuations of $A(p)$ or $A(q)$ (which potentially eliminates 2n sequence terms,

although usually many of these will already be removed from consideration), and, second, the solution sequence may be terminated as soon as $c(i) \geq c(p) + c(q)$. This knowledge can be exploited with the use of the g_k list as follows. When any $A(q)$ is generated, check the value g_k, where $A_o - A(q)$ is the kth element of the additive group. If $g_k = 0$, then nothing is to be done, but if $g_k = p > 0$, then $A(p) + A(q) \equiv A_o$. Thus set $\bar{d}_p = \bar{d}_q = 0$, so that no permissible continuation of $x(p)$ or $x(q)$ will be generated. In addition, if any continuation $x(i)$ of $x(p)$ (permissible or otherwise) has already been generated, we may set $\bar{d}_i = 0$. The fact that $x(p) + x(q)$ is a solution to (2) is recorded, unless some better feasible solution has already been found. Note that, although no continuation of certain of the sequence terms need be generated, these terms may still be used to drop other sequence terms in the usual fashion at instruction 2. The best feasible solution found supplies the value cx^* such that the sequence generation stops when $c(i) \geq cx^*$. (The process may also stop simply because T becomes empty.)

But we can generally do still better than this, extending the foregoing remarks to problem (3) in the process.

For each $x(k) \in \bar{S}_c$, let F_k be the set of those $x(i) \in \bar{S}_c$, $i \leq k$, such that $x(i) + x(k)$ is feasible for problem (3), and if $F_k \neq \emptyset$, let $x(v)$ be the particular element of F_k such that $v = \min(i: x(i) \in F_k)$.[1]

---

[1] If one is solving problem (2) subject to additional constraints, a modification of the g_k list can be conveniently used to identify F_k and $x(v)$. For this purpose, instead of assigning a single component g_k to the kth group element, one may define g_{k1} to be the index of the first $A(i)$ equivalent to the kth group element, g_{k2} to be the index of the second, and so on.

Then, between instructions 3 and 4 of the method we may insert the instruction:

3A. If $F_k = \emptyset$, go to instruction 4. Otherwise, set $\bar{d}_k = 0$ for all $x(i) \in F_k$, for $i = k$, and for each $x(i) \in \bar{S}_c$ that is a continuation of some solution in F_k. If $c(k) + c(v) < cx^*$, where x^* denotes the best feasible solution previously found, designate $x(k) + x(v)$ to be x^*.

When the method is augmented by instruction 3A, the condition $T = \emptyset$ of course may not indicate the absence of a feasible solution, but only that no solution exists that improves upon x^*, provided x^* is well-defined.

Proposition 6 assures that for each $x(k) \in \bar{S}$ there is an $x(i) \in \bar{S}$ such that $x(k) + x(i)$ is optimal if an optimal continuation of $x(k)$ exists. But then, because the $c(i)$ are monotonically nondecreasing, the first $x(i)$ such that $x(i) + x(k)$ is feasible must be optimal. The justification of 3A follows immediately, and the algorithm may clearly be halted at instruction 2 as soon as $N_J \geq cx^*$. We now state an important condition that allows the algorithm to be halted still earlier.

Proposition 7. When the method of Section 3 is augmented by instruction 3A, then the algorithm may be halted at instruction 2, and the best feasible solution x^* found taken as optimal, whenever $N_J - c(h) > c_M$, where $c_M = \text{Max}(c_j)$ and $c(h) = \max(c(i): c(i) < cx^* - N_J)$. (Until a feasible solution is found, let $c(h) = \infty$.)

Note: Proposition 7 may be interpreted as follows. Suppose a feasible solution x^* has been found by the method with instruction 3A included. Until such a solution is found, the method is the same as before. However, the next solution $x(r)$ to be generated,

and all remaining ones, must satisfy $c(r) \geq N_J$ for the successive current values of N_J. Now, if x^* is not optimal, and if there is any solution $x(h)$ such that $x(h) + x(r)$ is optimal, then $c(h)$ cannot exceed the value assigned to it by Proposition 7. Also, by thus assigning $c(h)$ its maximum possible value and $c(r)$ its minimum possible value N_J, one minimizes $c(r) - c(h)$ for all r and h such that $c(r) + c(h) < cx^*$, $c(r) \geq c(h)$. The proposition then asserts (through the relation $N_J - c(h) > c_M$) that once this minimum difference exceeds c_M, an optimal solution has already been found.

We break the proof of Proposition 7 into several parts, as follows.

Remark 8. Let $\bar{x}$ and $\hat{x}$ be chosen so that they minimize $c\bar{x} - c\hat{x}$ subject to $c\bar{x} - c\hat{x} \geq 0$, $\bar{x}, \hat{x} \geq 0$, and $\bar{x} + \hat{x}$ is an optimal solution to problem (3). Then $c\bar{x} - c\hat{x} \leq c_M$.

Proof. Suppose the conclusion is false. Select any j such that $\bar{x}_j > 0$ and let $x_j' = \bar{x}_j - 1$, $x_j'' = \hat{x}_j + 1$, with the remaining components of x' and x" equal respectively to the corresponding components of $\bar{x}$ and $\hat{x}$. We must have $cx' - cx'' < 0$ since $cx' - cx'' = c\bar{x} - c\hat{x} - 2c_j$, and $\bar{x}$ and $\hat{x}$ are assumed to minimize $c\bar{x} - c\hat{x} \geq 0$. But then, from $c\bar{x} - c\hat{x} > c_M$ we obtain $cx'' - cx' < 2c_j - c_M \leq c_M$, where x" and x' satisfy the requirements for $\bar{x}$ and $\hat{x}$, contrary to assumption.

Remark 9. Let q be the least index such that $x(q) \in \bar{S}$ and for some $p \leq q$, $x(q) + x(p)$ is optimal ($x(p) \in \bar{S}$). Then $c(q) - c(p) = c\bar{x} - c\hat{x}$ for $\bar{x}$ and $\hat{x}$ chosen as in Remark 8 and for all p, $p \leq q$, such that $x(p) + x(q)$ is optimal.

Proof. Suppose that for q and for some p as given, we have $c\bar{x} - c\hat{x}$

$< c(q) - c(p)$. Then from $c\bar{x} + c\hat{x} = c(q) + c(p)$ we obtain $c\bar{x} < c(q)$. By Proposition 6 there is a solution $x(s) \in \bar{S}$ such that $x(s) + \hat{x}$ is optimal, and also, therefore, a solution $x(r) \in \bar{S}$ such that $x(r) + x(s)$ is optimal. It is immediate that $c(s) = c\bar{x}$ and $c(r) = c\hat{x}$. But then $c(r) \leqq c(s) < c(q)$, implying $r, s < q$, which is impossible.

Proof of Proposition 7. Let S' denote the sequence generated when the method is augmented by instruction 3A. It may readily be verified that the proofs of Propositions 1, 2, 3 and 5 apply to S' as well as to $\bar{S}$, since these proofs do not depend upon the legitimacy of eliminating continuations of solutions in $\bar{S}_c$ from $\bar{S}$. (Only Proposition 6 depends upon such legitimacy.) Likewise, one may conclude from the reasoning of Proposition 5 that S' is a subsequence of $\bar{S}$. Consequently, we may assume that the indices of solutions in S' are assigned so that $x(i) \in S'$ and $x(i) \in \bar{S}$ denote the same solution. Then let q be selected as in Remark 9, and given this value of q, let p assume the smallest value ($p \leqq q$) such that $x(q) + x(p)$ is optimal. We now wish to establish that $x(p)$ and $x(q)$ belong to S'. If not, one or the other of them must be excluded from S' as a consequence of instruction 3A, and thus may be expressed in the form $x(i) + z$ or $x(k) + z$, where $x(i) + x(k)$ was discovered to be feasible, $x(i) \in F_k$, and $z \geqq 0$. Since $k \leqq q$ is required if $x(p)$ or $x(q)$ is to be eliminated as a consequence of finding the feasible solution $x(i) + x(k)$, we have $c(k) \leqq c(q)$. Then from $c(k) + c(i) \geqq c(q) + c(p)$ we also have $c(p) \geqq c(i)$, and hence $c(k) - c(i) \leqq c(q) - c(p)$. If this last holds as an equality, then $c(k) = c(q)$ and $c(p) = c(i)$, from which it follows by the definition of q that $k = q$, and $x(q) \in S'$. ($x(k)$ and $x(i)$ are not dropped

from S', but only their continuations not in S'. Of course, one may alternately, if desired, think of x(k) and x(i) as being dropped also.) But then, also $p \leq i$, hence x(p) cannot have been eliminated from S'. In particular, p = v for v as defined for instruction 3A, and hence the solution x(p) + x(q) would have been chosen as x* by the method. Thus, we examine the case for c(k) - c(i) < c(q) - c(p). By Remark 9 it follows that c(k) + c(i) > c(q) + c(p). Therefore c(p) < c(i), p < i, and x(p) cannot have been eliminated from S'. Thus, if $x(q) \notin S'$, or if x(q) + x(p) was not selected as x*, we must have q > k and either x(q) = x(k) + z or x(q) = x(i) + z. Suppose x(q) = x(k) + z. Then from c(q) + c(p) < c(i) + c(k) we obtain cz + c(p) < c(i). Since x(k) + z + x(p) is optimal, by Proposition 6 there is a solution $x(r) \in \bar{S}$ such that x(k) + x(r) is optimal, c(r) < c(i). But this is incompatible with q > k. One similarly obtains a contradiction from the assumption x(q) = x(i) + z, which completes the proof, for we have shown both that x(p), $x(q) \in S'$ and that x(q) + x(p) was at some point selected as x*.

To get an idea of the restrictiveness of the terminating condition of Proposition 7, we note that it allows the algorithm to stop whenever $N_J \geq cx^*/2 + c_M$ (where cx* is the optimal objective function value), and generally sooner. This is likely to result in considerable computational savings in the event that it is not difficult to check for feasibility and c_J is somewhat smaller than cx*/2. (Savings would likely result in any case due to the continuations that are dropped.) The reason for this lies in the fact that there are generally many more distinct solutions x that yield the same value of cx for cx large than for cx small. For example, if the objective is to minimize $\sum x_j$ and if cx* = 20, it is

evident that the number of solutions z that satisfy $0 \leq \sum z_j \leq 11$ is considerably smaller than those that satisfy $12 \leq \sum x_j \leq 20$. Of course, the method would ordinarily generate only a small fraction of the latter range of solutions in any event. But the existence of a cut-off point at $\sum x_j < 12$ would likely allow a further elimination of possibilities.

The question arises as to whether it is possible to shortcut the generation of solutions if it is known beforehand, or explicitly required, that an optimal solution must satisfy a constraint of the form $\sum x_j \geq L$. We see that this question may be answered affirmatively in the next section.

7. Accommodating $\sum x_j = L$, $\sum x_j \geq L$, and related constraints.

We have already seen how to handle constraints of the form $\sum x_j \leq M$. The method for handling constraints of the form $\sum x_j = L$ and $\sum x_j \geq L$ is quite similar. Suppose that $c_1 = \text{Min}\,(c_j)$. To accommodate $\sum x_j = L$, x_1 is eliminated from the objective function by pivot reduction (i.e., by replacing x_1 by $L - \sum_{j \geq 2} x_j$), so that the new objective function coefficient for each j is $c_j' = c_j - c_1 \geq 0$. (If $c_j' = 0$ for some $j \neq 1$, then the problem is of course perturbed so that $c_j' > 0$.) With x_1 thus "eliminated," only the variables x_j for $j \geq 2$ enter explicitly into consideration, and $\sum x_j = L$ with $x_1 \geq 0$ is accomodated by requiring $\sum_{j \geq 2} x_j \leq L$.

The constraint $\sum x_j \geq L$ is handled in a like fashion, introducing the nonnegative integer slack s_1 and writing $\sum x_j - s_1 = L$. After removing x_1 from the objective function as above, and treating it as

dependent variable (s_1 now has the objective function coefficient c_1), we are left with the inequality $\sum_{j\geq 2} x_j - s_1 \leq L$. To assure that the method will actually generate only those solutions satisfying $\sum x_j \geq L$, it is important that s_1 be indexed as the last x variable; i.e., $s_1 = x_{n+1}$. As a consequence, whenever $\sum_{j=2}^{n} x_j^i - x_{n+1}^i = L$, no permissible continuation of x(i) may be allowed, since every permissible continuation of this solution would necessarily violate $\sum_{j=2}^{n} x_j - x_{n+1} \leq L$. Note that this would not be the case if s_1 were not the last x variable.

Variations immediately suggest themselves. Whenever the problem has a subset of constraints of the form $\sum a_{ij}x_j \leq a_{io}$, where, say, $a_{ij} \geq 0$ for $j = 1, \ldots, r_i$ and $a_{ij} \leq 0$ for $j = r_i + 1, \ldots, n$, then it can readily be seen how to apply the algorithm so that every x(i) generated satisfies all such constraints for which $a_{io} \geq 0$.[1] In accordance with these remarks, it would be reasonable to attempt to impose an indexing of variables that would create this structure, or nearly this structure, for several of the problem constraints. For constraints that could not be put in this form (given that other constraints do have the desired form), it would be useful to index as many positive a_{ij} as possible ahead of the first negative a_{ij}.

---

[1] Similarly, for constraints of the form $\sum a_j x_j \equiv a_o \pmod 1$, where the a_j may represent either scalars or column vectors, if the a_j for $j = 1, \ldots, r$, are contained in a subgroup of the group generated by the $a_j \pmod 1$ for all j (e.g., $a_j \equiv 0$ for $j \leq r$), then if $a_o - \sum a_j x_j^i$ is not a member of the subgroup, the continuations $x(i) + e_j$ for $j \leq r$ need not be generated. To exploit this and similar relations more thoroughly, it is of course possible to reindex the x_j for the permissible continuations of x(i). However, such local reindexing requires more memory, and does not permit a solution x(i) to be dropped unless $\bar{R}^*(x(i)) = \emptyset$, or unless $\bar{R}^*(x(i)) \neq \emptyset$ can be shown to imply $\bar{R}^*(x(p)) \neq \emptyset$ for some solution x(p) not dropped.

8. Passive Variables and Additional Ways of Handling Upper Bounds for Problem 2.

We will be concerned in this section with two additional ways of handling certain types of problem constraints that arise chiefly in the context of problem (2). The first way involves the creation of "passive" variables to restrict the number of solutions generated. The second involves a means for determining when some subset of exogenous bounds is actually nonbinding, thus allowing the bounds to be treated as natural. To accomplish the second goal we will subsequently derive some needed results about finite additive groups.

Suppose first that $x_j \leq B_j$ is required for all x_j, and that there exist nonnegative integers h, k ($h > 0$) such that $kc_1 \leq hc_2$, $kA_1 \cong hA_2$. Ordinarily, if ke_1 is generated as x(p) and he_2 is generated as x(q), q > p, it would not be permissible to drop x(q) due to the existence of the exogenous B_1. However, note that we may require of any optimal solution that $x_2 \geq h$ only if $x_1 > B_1 - k$ (equivalently, $x_1 \leq B_1 - k$ only if $x_2 \leq h - 1$). For if $x_2 \geq h$ and $x_1 \leq B_1 - k$, one may obtain a solution with no greater value of cx by decrementing x_2 by h and incrementing x_1 by k. To take advantage of this fact we create the "passive" variable $x_{1,2} = hx_2 + (B_1 - k + 1)x_1$. We call $x_{1,2}$ passive because when $x_{1,2}$ is generated as a solution x(i),[1] we remove $x_{1,2}$ from the list of variables, and assign a large enough value to $\bar{d}_i$ so that all original variables may combine with x(i) to form other solutions. In doing this, we restrict x_2 so that $x_2 \leq h - 1$, except for

---

[1] By the obvious convention, we mean by this that $x_1^i = B_i - K + 1$, $x_2^1 = h$, and $x_j^i = 0$ for $J \geq 3$.

the continuations of x(i).[1]

There are other, more general, instances in which the creation of passive variables may be useful. We will not attempt to give a formal description of these, but instead provide some examples from which the general procedures may be inferred. Suppose, for example, that by a relationship such as outlined above, or by some other means, it is known that $x_2 \geq 5$ implies $x_1 \geq 3$ and that $x_3 \geq 7$ implies $x_2 \geq 6$. To handle this we may create the passive variables $x_{1,2} = 3x_1 + 5x_2$ and $x_{1,2,3} = 3x_1 + 6x_2 + 7x_3$. Then we impose the restrictions $x_2 \leq 4$, $x_3 \leq 6$, except that $x_2 \leq 1$ when x_2 combines with $x_{1,2}$, and $x_2 \leq B_2 - 6$, $x_3 \leq B_3 - 7$ when variables combine with $x_{1,2,3}$. If, in addition to the foregoing, $x_4 \geq 5$ implies $x_3 \geq 2$, we create the <u>two</u> additional passive variables $x_{3,4} = 2x_3 + 5x_4$ and $x_{1,2,3,4} = x_{1,2,3} + 5x_4$. The way in which such passive variables may be created to accommodate other similar situations should by now be clear.[2]

---

[1] Note that we could alternately let $x_{1,2} = (B_1 - k + 1)x_1$ (hence $x_1^i = B_1 - k + 1$, $x_j^i = 0$ for $j \geq 2$) and then require $x_1 \leq B_1 - k$ and $x_2 \leq h - 1$ except for continuations of x(i).

[2] Passive variables provide a base upon which the rules of the algorithm generate a partitioning of the x(i). The justification for the partitioning may be expressed in terms of logical equivalences; e.g., in a simple case: $(P \rightarrow Q) \Longleftrightarrow (P \wedge Q) \vee (\sim P)$. An interesting special use of the passive variables occurs when the logical alternatives provide lower bounds on cx, which may then be evaluated in place of $\bar{N}_j$ for such variables. Passive variables may also, of course, be created at later stages of the solution process. Thus, if it is determined that the permissible continuations of x(i) must satisfy $x_j \geq L_j$, where $L_j > x_j^i$ for some subset R of the $j \leq d_i$, then one may take advantage of this by creating x_u, say, where generating x_u as a solution x(p) gives $x_j^p = L_j$ for $j \in R$ and $x_j^p = x_j^i$ otherwise. In this case, d_p is set equal to d_i, and x(i) may be disregarded as a source of continuations.

We now turn to the second means of handling upper bound restrictions. It may well happen that certain exogenous B_j do not in reality exert any influence upon the set of optimal solutions to (2). We derive some results concerning additive groups that will help to determine when this is the case.

Imagine, in particular, that one has found k, h such that $hc_1 \leq kc_2$ and $hA_1 \equiv kA_2$. Under some circumstances, it is possible to impose the restriction $x_2 \leq k - 1$, using $k - 1$ as a natural bound for x_2, in spite of the fact that $x_1 \leq B_1$ for B_1 exogenous. We develop one such circumstance as follows.

Proposition 8. Assume $z \in G$ if and only if $z \equiv \sum A_j x_j$, $B_j \geq x_j \geq 0$, and x_j integer. Let $h_1A_1 \equiv k_1A_2$ and $h_2A_1 \equiv k_2A_2$, where $h_1 \leq h_2$ and $B_1 + 1 \geq h_2$, $B_2 + 1 \geq k_1$ (h_1,h_2,k_1,k_2 nonnegative integer). Then

(i) $k_1h_2 > k_2h_1$ implies

(ii) $z \in G$ if and only if $z \equiv \sum A_j x_j$, x_j integer, $B_j \geq x_j \geq 0$ for $j \geq 3$ and $h_2 - 1 \geq x_1 \geq 0$, $k_1 - 1 \geq x_2 \geq 0$.

Proof. Assume $z \equiv \sum A_j \bar{x}_j$, $\bar{x}_j$ integer, $B_j \geq \bar{x}_j \geq 0$ but $\bar{x}_2 \geq k_1$ or $\bar{x}_1 \geq h_2$. If $\bar{x}_2 \geq k_1$, let $x_1' = \bar{x}_1 + h_1$, $x_2' = \bar{x}_2 - k_1$, and $x_j' = \bar{x}_j$ for $j \geq 3$. Then we have $z \equiv \sum A_j x_j'$. Consider the linear function $L(x) = k_2x_1 + h_2x_2$. By (i) it follows that $L(x') < L(\bar{x})$. We replace x' by $\bar{x}$ and repeat the process with $L(\bar{x})$ strictly decreasing at each step, until $\bar{x}_2 \leq k_1 - 1$. Now possibly $\bar{x}_1 \geq h_2$ (or possibly $\bar{x}_2 < k_1$ and $\bar{x}_1 \geq h_2$ to begin with). Define $x_1' = \bar{x}_1 - h_2$, $x_2' = \bar{x}_2 + k_2$, $x_j' = \bar{x}_j$ for $j \geq 3$, so that, again $z \equiv \sum A_j x_j'$. In this case $L(x') \leq L(\bar{x})$. Denoting x' by $\bar{x}$ and repeating, eventually $\bar{x}_1 \leq h_2 - 1$. If now $\bar{x}_2 \geq k_1$, the original replacement process is initiated. Continuing this cycle of replacing $\bar{x}$ by x' we see that

$L(\bar{x})$ is always nonincreasing and is periodically strictly decreasing. From (i) it follows that $h_2 > 0$ and thus (ii) is true or else eventually $\bar{x}_2 < 0$. But the latter is impossible by the way in which x' is defined in terms of $\bar{x}$.

To see specifically how Proposition 8 relates to the problem of determining natural bounds, suppose that no exogenous bounds B_j existed and that h_1e_1 was generated as a solution $x(p)$. Then, if subsequently the solution k_1e_2 was considered for inclusion among the $x(i)$, where $k_1A_2 \equiv h_1A_1$ (as in Proposition 8), the fact that $h_1c_1 \leqq k_2c_2$ would permit the solution k_1e_2 to be dropped and the restriction $x_2 \leqq k_1 - 1$ to be imposed. Likewise, the method would similarly permit $x_1 \leqq h_2 - 1$ to be imposed if k_2e_2 were generated as $x(p)$ and $k_2A_2 \equiv h_2A_1$, $k_2c_2 \leqq h_2c_1$.

Observe now that $h_1c_1 \leqq k_1c_2$ and $k_2c_2 \leqq h_2c_1$ imply $k_2h_1 \leqq k_1h_2$. This follows by multiplying the first inequality by h_2, the second by h_1, combining, and then dividing through by $c_2 > 0$. But by Proposition 8, if in fact $k_2h_1 < k_1h_2$, then imposing $x_1 \leqq h_2 - 1$ and $x_2 \leqq k_1 - 1$ will be permissible even if $x_j \leqq B_j$ for exogenous B_j (provided the new bounds for x_1 and x_2 do not exceed B_1 or B_2).

Suppose now that the relation between x_1 and x_2 stated in Proposition 8 also holds between other (not necessarily disjoint) pairs of variables. Creating the appropriate linear function $L(x)$ for each such pair affords the conclusion that x_j may be bounded from above by the smallest of the upper bounds that apply by Proposition 8, provided this smallest upper bound is $\leqq B_j$. We now seek an expedient way to determine the smallest of these bounds for each j.

Proposition 9. Let k_1 be the least positive integer such that, for some integer $h \geqq 0$, $k_1A_2 \equiv hA_1$ and $k_1c_2 \geqq hc_1$. Moreover, let h_1

be the smallest integer h satisfying these relations. Similarly, let h_2 be the least positive integer such that, for some integer $k \geq 0$, $h_2A_1 \equiv kA_2$ and $h_2c_1 > kc_2$, and let k_2 be the smallest k satisfying these relations. (Alternately, we may require $k_1c_2 > h_1c_1$ and $h_2c_1 \geq k_2c_2$.) Then $h_1k_2 < h_2k_1$ and, moreover, $h_1 < h_2$, $k_2 < k_1$.

Proof. First, observe that $h_1 \geq h_2$ and $K_1 \leq k_2$ is impossible. For otherwise we have $k_2c_2 < h_2c_1 \leq h_1c_1 \leq k_1c_2 \leq k_2c_2$, or $k_2c_2 < k_2c_2$. Thus, either $h_1 < h_2$ or $k_2 < k_1$. Suppose $h_1 < h_2$ but $k_2 \geq k_1$. Then $(k_2 - k_1)c_2 < (h_2 - h_1)c_1$ and $(k_2 - k_1)A_2 \equiv (h_2 - h_1)A_1$, where $k_2 > k_2 - k_1 \geq 0$, $h_2 \geq h_2 - h_1 > 0$. But then by the definition of h_2 we have $h_2 - h_1 = h_2$ and thus $k_2 - k_1 = k_2$, the latter being impossible. Consequently, $h_1 < h_2$ implies $k_2 < k_1$. On the other hand, if $k_2 < k_1$ and $h_1 \geq h_2$, then $(k_1 - k_2)c_2 > (h_1 - h_2)c_1$ and $(k_1 - k_2)A_2 \equiv (h_1 - h_2)A_1$. In this case we obtain a contradiction by observing that $h_1 > h_1 - h_2 = h_1$ follows from the definition of k_1 and h_1. Thus we conclude $h_1 < h_2$, $k_2 < k_1$, and, of course, $h_1k_2 < h_2k_1$.

One immediate consequence of Proposition 9 is that for h_1, h_2, k_1, k_2 as indicated, it is impossible to find h, k not both zero such that $h_2 > h \geq 0$, $k_1 > k \geq 0$ and $hA_1 \equiv kA_2$. Moreover, h_1, h_2, k_1, k_2 may always be determined to satisfy the conditions of Proposition 9 as long as c_1, $c_2 > 0$. This means that we are provided with natural bounds $\bar{B}_1 = h_2 - 1$ and $\bar{B}_2 = k_1 - 1$ such that $\bar{B}_1 + \bar{B}_2 < D$, where D is the order of the additive group generated by the A_j. (We assume that if exogenous bounds B_1 and B_2 exist, then $\bar{B}_1 \leq B_1$, $\bar{B}_2 \leq B_2$.) This result extends to any number of variables, since if $hA_r \equiv kA_s$ for nonnegative h and k not both 0,

it is immediate by Propositions 8 and 9 that either $x_r \leq h - 1$ or $x_s \leq k - 1$ can be enforced. By a process of scanning, then, one selects bounds $\bar{B}_j$ such that $x_j \leq \bar{B}_j$ and $\sum \bar{B}_j < D$, provided $\bar{B}_j \leq B_j$. Gomory has pointed out that $\sum x_j \leq D - 1$ may be enforced for an optimal solution to (2). We have here indicated a way to determine bounds on the x_j that satisfy this inequality (provided exogenous bounds are nonbinding). One of the chief values of this result, however, lies in the fact that it provides a convenient way to check whether some of the exogenous B_j may actually be replaced by natural (and possibly smaller) upper bounds.

Another use for Propositions 8 and 9 occurs when $\sum x_j$ is bounded above. In this event, the upper bounds $x_1 \leq h_2 - 1$, $x_2 \leq k_1 - 1$ still apply provided $h_1 \leq k_1$ and $k_2 \leq h_2$. It is to be noted that these latter inequalities may occur for h_1, h_2, k_1, k_2 satisfying Proposition 8 but not Proposition 9.

In the scanning of the A_j and their multiples to determine bounds on the x_j, there is another result that may be useful. If h_1 is the least positive (integer) multiple of A_1 such that $h_1A_1 \equiv kA_2$ for some k, then the only positive multiples h of A_1 that satisfy $hA_1 \equiv kA_2$, for some k, are $h = h_1, 2h_1, 3h_1$, etc. To see this, suppose $hA_1 \equiv kA_2$ for $h = rh_1 + q$, where r is any positive multiple and $0 \leq q < h_1$. Then $qA_1 \equiv (k - rk_1)A_2$, where $k_1A_2 \equiv h_1A_1$. By the definition of h_1 it follows that $q = 0$.

9. Concluding Remarks

The chief focus of this paper has been on developing an algorithm for solving problem (2) when certain additional constraints apply. Gomory's approach [5] to solving this problem in the absence of such constraints is to use the dynamic programming recursion

$$\phi_j(z) = \text{Min}(\phi_{j-1}(z),\ \phi_j(z - A_j) + c_j), \qquad j = 1, \ldots, n,$$

where $\phi_j(0) = 0$, $\phi_o(z) = \infty$, and z ranges over the elements of the additive group generated by the A_j mod 1. If $\phi_j(z - A_j)$ is unknown, then one may provisionally replaced it by $\phi_{j-1}(z - A_j)$ and be assured that computing $\phi_j(z + kA_j)$ based upon this replacement for $k = 0, 1, \ldots,$ will yield the correct value of $\phi_j(z - A_j)$ for the value of k such that $z + kA_j \equiv z - A_j$.

As W. W. White points out [6], it is possible to determine $\phi(z + kA_j)$ correctly for all k without the need for revision. To do this one identifies h such that $\phi_{j-1}(z + hA_j) = \min_k(\phi_{j-1}(z + kA_j))$, and defines $\phi_j(z + hA_j) = \phi_{j-1}(z + hA_j)$. $\phi_n(A_o)$ then gives the optimal objective function value for (2). Backtracking over the $\phi_n(z)$ to find an optimal x occurs in a manner related to that outlined in Section 5 by recording for each $\phi_j(z)$ the largest k such that $x_k = 1$.

Our motivation in developing the algorithm of this paper has not been to devise a method that is competitive in efficiency with Gomory's when applied to (2) in the absence of additional constraints. Nevertheless, our method may be competitive in this limited context due to the fact that it may generate A_o somewhat in advance of generating all the other elements of the finite additive group, particularly if the variation of Section 6 is used. The Gomory approach, on the other hand, must generate $\phi_j(z)$ for each z in the group and for each j except possibly for $j = n$. It may be noted that the values $\phi_n(z)$ constitute a superset of the c(i) generated by the algorithm of this paper.

Once problem (2) becomes complicated with restrictions of the form $x_j \leq B_j$, however, the dynamic programming recursion just outlined no longer suffices, and it appears necessary to resort to the more familiar Bellman knapsack recursion

$$\phi_j(z) = \underset{x_j}{\text{Min}} \left\{ \phi_{j-1}(z - A_j x_j) + c_j x_j \mid 0 \leq x_j \leq B_j \right\}.$$

In addition to being rather demanding on memory capacity, it is evident that this approach can require considerably more computation than the earlier recursion unless the average of the B_j is not too far from 1.

The treatment of exogenous upper bounds in our approach requires, by comparison, very moderate storage capacity. In addition, although these bounds may sometimes entail more computation than would be required in their absence, they may also sometimes entail less. This is due to the fact that their existence may rule out the generation of certain x(i) that would otherwise occur in the solution sequence.

There appears to be little promise in the customary dynamic programming approach to handling other kinds of restrictions, since the amount of computation and memory requirements in such applications are typically quite large.

We might note, in passing, that our method can be employed in the framework of a cutting approach. The optimal solution to (2) -- or to (2) augmented by some of the restrictions of (1) -- provides a lower bound L such that $cx \geq L$ in the optimal solution to (1). One may find successively larger integer values for L by solving (1) as a linear program with the constraint $cx \geq L$ adjoined, and then reapplying the method of this paper. The method may clearly be adapted to solve for all optimal solutions to (2) or (3), thereby assuring that L may always be incremented by at least 1 (assuming that c initially consists of integers).

More generally, any linear form in the nonbasic variables with positive coefficients (and possibly some zero coefficients temporarily perturbed), can replace the objective function of (3). The optimal value for this modified objective function -- or a lower bound on this optimal

value obtained by the method at some convenient cutoff point -- can be used to transform the linear form directly into a cut.[1] If the problem has a subset of constraints of the form $\sum a_j x_j \equiv a_o$, where $a_j \equiv 0$ for $j \leq r$, then a cut can also be obtained by minimizing a linear form over the x_j for $j > r$, provided all constraints $\sum a_j x_j \equiv a_o$ with $a_j \not\equiv 0$ for $j \leq r$ are disregarded.[2] It may be noted that the variation of Section 6 may provide a particularly effective way to determine a cut. If an optimal solution hasn't been found upon reaching a specified cutoff, it follows from Proposition 7 that one may impose the cut $\sum c_j x_j \geq \text{Max}(c(i), 2c(i) - c_M)$, where $x(i)$ is the last solution generated. When the linear form minimized is $\sum x_j$, this of course becomes $\sum x_j \geq 2 \sum x_j^i - 1$.

---

[1] It is assumed that the coefficients of the linear form are selected so that the slack variable of the cut will be integer valued.

[2] One can take advantage of these latter constraints, however, by following the approach of the footnote on page 32.

References

1. Dantzig, G. B., "On the Shortest Route Through A Network," Management Science, Vol. 6, No. 2, 1960.

2. Gilmore, P. C., and R. E. Gomory, "Multi-Stage Cutting Stock Problems of Two and More Dimensions," Operations Research, Vol. 13, 1965.

3. Glover, Fred, "Truncated Enumeration Methods for Solving Pure and Mixed Integer Linear Programs," Working Paper, Operations Research Center, University of California, Berkeley, May 1966.

4. Gomory, R. E., "An Algorithm for Integer Solutions to Linear Programs," in R. L. Graves and P. Wolfe, eds., Recent Advances in Mathematical Programming, McGraw-Hill Book Company, Inc., 1963.

5. Gomory, R. E., "On the Relation Between Integer and Noninteger Solutions to Linear Programs," Proceedings of the National Academy of Sciences, Vol. 53, 1965.

6. White, William W., "Integer Linear Programming: Relations Between Discrete and Continuous Solutions," ORC 66-27, 1966.

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

University of California, Berkeley

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3. REPORT TITLE

AN ALGORITHM FOR SOLVING THE LINEAR INTEGER PROGRAMMING PROBLEM OVER A FINITE ADDITIVE GROUP, WITH EXTENSIONS TO SOLVING GENERAL LINEAR AND CERTAIN NONLINEAR INTEGER PROGRAMS

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Research Report

5. AUTHOR(S) *(Last name, first name, initial)*

Glover, Fred

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
September 1966	46	6

8a. CONTRACT OR GRANT NO.

Nonr-222(83)

b. PROJECT NO.

NR 047 033

c.

Research Project No. RR 003-07-01

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

ORC 66-29

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES

Available without limitation on dissemination

11. SUPPLEMENTARY NOTES

See bottom of title page

12. SPONSORING MILITARY ACTIVITY

Mathematical Science Division

13. ABSTRACT

See Abstract Page

DD FORM 1 JAN 64 1473

Unclassified
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Integer Programming						
Truncated Enumeration						
Gomory Additive Group						
"Least Cost" Algorithm						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2*a*. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2*b*. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7*a*. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7*b*. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8*a*. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8*b*, 8*c*, & 8*d*. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9*a*. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9*b*. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS), (S), (C), or (U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.

DD FORM 1 JAN 64 1473 (BACK)