Cambridge Scientific Center

36. Y03
July 1966

IBM
Data Processing Division

Application of Combinatorial Programming to a Class of All-Zero-One Integer Programming Problems

36. Y03

APPLICATION OF COMBINATORIAL PROGRAMMING TO A CLASS OF ALL-ZERO-ONE INTEGER PROGRAMMING PROBLEMS

J. F. Pierce

IBM Cambridge Scientific Center Report

International Business Machines Corporation
Cambridge Scientific Center
Cambridge, Massachusetts

July, 1966

36.Y03
July, 1966
Scientific Center Report
Limited Distribution

APPLICATION OF COMBINATORIAL PROGRAMMING TO A CLASS OF ALL-ZERO-ONE INTEGER PROGRAMMING PROBLEMS

J. F. Pierce

International Business Machines Corporation
Cambridge Scientific Center
Cambridge, Massachusetts

Abstract

Problem-solving procedures based on the methods of combinatorial programming are presented for solving a class of integer programming problems in which all elements are zero or one. All of the procedures seek first a feasible solution and then successively better and better feasible solutions until ultimately one is discovered which is shown to be optimal. By representing the problem elements in a binary computer as bits in a word and employing logical "and" and "or" operations in the problem-solving process, a number of problems involving several hundred integer variables have been solved in a matter of seconds.

Index Terms for the IBM Subject Index

Operations Research
Combinatorial Programming
Integer Programming
05-Computer Application
13-Management Sciences

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication elsewhere and has been issued as a Technical Report for early dissemination of its contents. As a courtesy to the intended publisher, it should not be widely distributed until after the date of outside publication.

TABLE OF CONTENTS

I. INTRODUCTION

In this paper are considered some direct algorithms for solving integer programming problems of the form:

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

$$\text{Subject to:} \quad \sum_{j=1}^{j=n} x_j \underline{A}_j = \underline{R} \qquad (1)$$

$$\text{and} \quad x_j = 0 \text{ or } 1 \qquad j = 1, 2, \ldots, n$$

and of the form:

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

$$\text{Subject to:} \quad \sum_{j=1}^{j=n} x_j \underline{A}_j \geq \underline{R} \qquad (1')$$

$$\text{and} \quad x_j = 0 \text{ or } 1 \qquad j = 1, 2, \ldots, n$$

where $\underline{A}_j$ is an mxl column vector $(a_{1j}, a_{2j}, \ldots, a_{mj})$, each element of which is zero or one, and $\underline{R}$ is an mxl column vector $(r_1, r_2, \ldots, r_m)$ in which all elements are one. All c_j are assumed to be nonnegative.

As discussed by Balinski [3], problems which can be put in these forms arise in such diverse contexts, for example, as that of the engineer in designing switching circuits, the logician in seeking a simplest disjunctive normal form which is equivalent to a given formula[1], and the business man in distributing goods from a central warehouse to a group of outlying clients. To this diversity might be added that of the

---

[1] For application in the operations research field see, for instance, the article by Root [18].

industrial engineer in balancing an assembly line [20], the PERT-CPM project manager in selecting from among feasible alternatives the best means for accomplishing particular tasks [5], and the computer systems analyst in retrieving information most efficiently from a set of files [7].

The methods investigated for solving these problems are those of combinatorial programming. By combinatorial programming, we mean, after Rossman and Twery [19] problem-solving procedures based on: (i) the use of a controlled enumerative procedure for considering (at least implicitly) all potential solutions; and (ii) the elimination from explicit consideration of particular potential solutions which are known from dominance, bounding and feasibility considerations to be unacceptable. We limit the meaning of combinatorial programs to procedures based on these two concepts which are reliable in the respect that when carried out to completion they guarantee the discovery of an acceptable solution (in the present problem, an optimal feasible solution) when one exists. Alternately, the methods to be employed might be termed "branch and bound" after Little et al [13] or, perhaps, "reliable heuristic programming".

While in general there are several basic search strategies that might be employed in combinatorial programs, principal attention will be focussed in the present paper on one wherein search is directed first to the discovery of a feasible solution and then to successively better and better feasible solutions until ultimately one is discovered which is shown to be optimal. The importance of this feature of course lies in the possibility of terminating problem-solving prematurely in time-consuming

problems with a useable although not necessarily optimal solution.

First, attention is focussed on problems of form (1). In the following section the basic algorithm is described and illustrated by means of a small example. While completely general, the algorithm has been developed to capitalize on the fact that since all elements are either zero or one they may be represented in a binary computer as bits in a word and operated on with logical "and" and "or" operations. Thus, in a problem with m constraints each column vector $\underline{A}_j$ can be represented as $\langle m/B + .99\ldots9 \rangle$ binary words, where B is the number of bits per word in the computer and $\langle \theta \rangle$ denotes the largest integer contained in θ. In Section III computational experience with the algorithm is discussed as obtained on an IBM 7094, a binary computer with $B = 36$. In Section IV several prospective modifications to the algorithm are noted, and finally, in Section V the extension to problems of the form (1') is outlined.

In relationship to other algorithms the present procedures appear most closely akin to the additive algorithm proposed by Balas [1] for solving the general zero-one integer programming problem. In the present procedures the underlying enumeration scheme is simpler, employing throughout problem-solving a fixed, prescribed preordering of the column vectors $\underline{A}_1, \underline{A}_2, \ldots, \underline{A}_n$. As a consequence relatively little problem-solving time is expended on matters of procedural planning and control, e.g., on performing bookkeeping tasks to record the portions of the tree elaborated and evaluated so far, on selecting the next variable x_j to consider, etc. With this simplicity and the manipulative economies achieved through representation of vectors as binary words, it has been

possible to solve a number of problems involving several hundred variables in a matter of seconds.

II. THE BASIC ALGORITHM

As has been mentioned, a first principle of combinatorial programming is that of controlled enumeration of potential solutions. To insure reliability of the resulting problem-solving procedure this implies that all 2^n of the possibilities $\underline{X} = (x_1, x_2, \ldots, x_n)$ must be considered, at least implicitly. In the present case the underlying enumeration scheme used to accomplish this is an elementary tree search method. If we define a potential solution $\underline{X}''$ to be lexicographically smaller than X' when $x_1'' = x_1'$; $x_2'' = x_2'$; ...; $x_{r-1}'' = x_{r-1}'$; and $x_r'' < x_r'$ for some r , $1 \leq r \leq n$, then the underlying scheme consists simply in enumerating all 2^n possibilities in lexicographically decreasing order, where the ordering of the column vectors $\underline{A}_1, \underline{A}_2, \ldots, \underline{A}_n$ remains fixed throughout problem-solving. As suggested in Figure 1, these possibilities can be conveniently represented in a tree structure where the j^{th} level of branches represents the values for x_j and where each node at the terminus of a branch at level j represents the subset of all potential solutions $\underline{X} = (x_1, x_2, \ldots, x_j, \ldots, x_n)$ in which $x_1 = \bar{x}_1$; $x_2 = \bar{x}_2$; ...; $x_j = \bar{x}_j$, the values of the $\bar{x}_i$ being determined by the path connecting the node with the origin. With the branches emanating from each node ordered with $x_j = 0$ on the right, the enumeration of potential solutions in lexicographically decreasing order corresponds to elaborating paths in the tree from left to right. Equivalently, therefore, enumeration may be viewed as entailing the successive stage by stage specification of a value for $x_1, x_2, \ldots, x_r$ until a complete combination $(x_1, x_2, \ldots, x_n)$ is

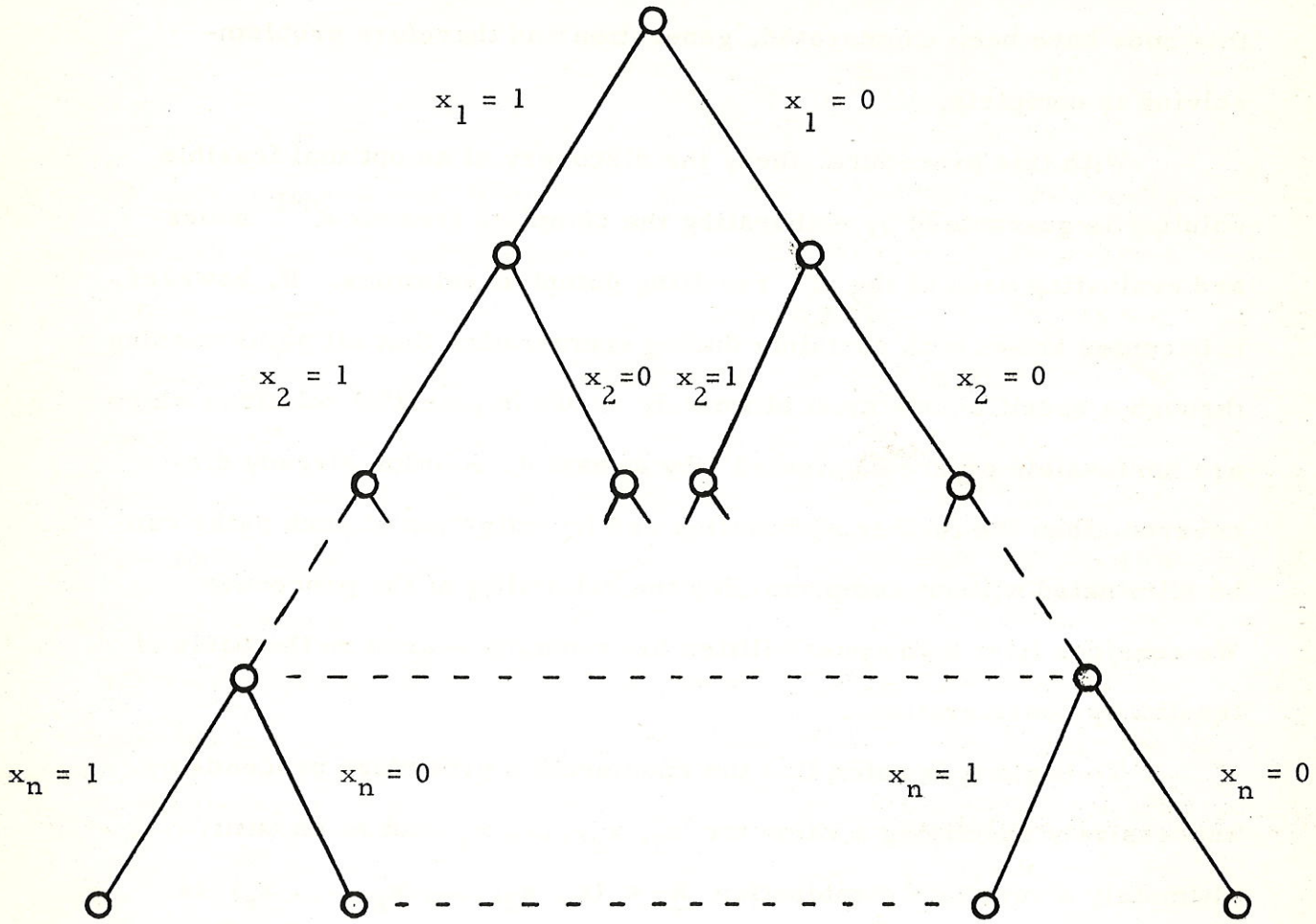

Figure 1. Tree representation of potential solutions as generated by the basic enumeration procedure.

determined, or as entailing the successive level by level selection of branches in the tree, one branch per level, until a terminal node is reached at level n. Upon reaching a terminal node the corresponding potential solution is evaluated and then the tree-elaboration process backtracks to the first node in the path for which both branches have not been enumerated and resumes with the second branch. When the process

has backtracked to the origin node and both branches emanating from this node have been enumerated, generation and therefore problem-solving is complete.

With this procedure, then, the discovery of an optimal feasible solution is guaranteed by elaborating the complete tree of 2^{n+1} nodes and evaluating each of the 2^n resulting potential solutions. If, however, it becomes known with certainty during enumeration that all paths passing through a specific node must ultimately result in potential solutions which are nonfeasible or are dominated[2] by a feasible solution already discovered, then the further elaboration and investigation of such paths can be eliminated without compromising the reliability of the procedure. We consider first some possibilities for reducing search on the basis of feasibility considerations.

To begin it is noted that the enumeration procedure proceeds by successively specifying a value for $x_1, x_2, \ldots, x_s$ and so on until ultimately a complete combination $\underline{X} = (x_1, x_2, \ldots, x_s, \ldots, x_n)$ is specified. At any stage s-1, having specified the values $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{s-1}$, there remains a problem of the same form as (1):

$$\text{Minimize} \quad \sum_{j=s}^{j=n} c_j x_j$$

$$\text{Subject to:} \quad \sum_{j=s}^{j=n} x_j \underline{A}_j = \underline{R} - \sum_{j=1}^{j=s-1} \bar{x}_j \underline{A}_j \qquad (2)$$

$$\text{and} \quad x_j = 0 \text{ or } 1 \qquad j = s, s+1, \ldots, n$$

Denoting by $\underline{R}_s = (r_1^s, r_2^s, \ldots, r_m^s)^T$ the column vector $\underline{R} - \sum_{j=1}^{j=s-1} \bar{x}_j \underline{A}_j$ which results upon the completion of stage s-1, it follows from the nonnegativity of the elements a_{ij} that for each k, $r_k^s \geq r_k^{s+1} \geq \ldots \geq r_k^n$

---

[2] Solution $\underline{X}$ will be said to dominate $\underline{X}'$ if $\sum_j c_j x_j \leq \sum c_j x'_j$.

for all permissible values of $x_s, x_{s+1}, \ldots, x_n$. Hence in specifying x_s if $r_k^s = 0$ for any k such that $a_{ks} = 1$, feasibility of (2) requires that $\bar{x}_s = 0$. In such a case none of the $2^{n+s+1} - 1$ nodes in the tree stemming from the branch $\bar{x}_s = 1$ need be explicitly considered.

Incorporating this feasibility consideration into the enumeration scheme the resulting procedure may be stated as follows:

(i) Set $x_1 = F(\underline{R}_1); x_2 = F(\underline{R}_2); \ldots; x_n = F(\underline{R}_n)$.

If $\underline{R} = \sum_{j=1}^{j=n} x_j \underline{A}_j$, save $\underline{X}$ as the best feasible solution $\underline{X}^0$ discovered so far, setting $Z^0 = \sum_{j=1}^{j=n} c_j x_j^0$. Go to (ii).

(ii) If there is a j, $1 \leq j < n$ such that $x_j = 1$, let s be the largest such j and go to (iii). Otherwise problem-solving is complete.

(iii) Redefine $x_1 = x_1$; $x_2 = x_2$; $\ldots$; $x_{s-1} = x_{s-1}$; $x_s = 0$; $x_{s+1} = F(\underline{R}_{s+1}); \ldots; x_n = F(\underline{R}_n)$. If $\underline{R} = \sum_{j=1}^{j=n} x_j \underline{A}_j$ and $\sum_{j=1}^{j=n} c_j x_j < Z^0$ save $\underline{X}$ as the best feasible solution discovered so far. Go to (ii).

where

$$\underline{R}_s = \underline{R} - \sum_{j=1}^{j=s-1} x_j \underline{A}_j$$

and

$$F(\underline{R}_s) = \begin{cases} 1 \text{ if } \underline{R}_s \otimes \underline{A}_s = \underline{A}_s \\ 0 \text{ otherwise .} \end{cases}$$

the symbol $\otimes$ denoting the logical "and" operation[3].

Another feasibility consideration arises from the fact that a necessary condition for the existence of a feasible solution to (2) at stage s is that for all k such that $r_k^s = 1$ there must exist at least one vector $\underline{A}_t$, $s \leq t < n$ such that $a_{kt} = 1$. One simple but effective means of employing this consideration is to pre-order[4] the vectors $\underline{A}_j$ by sets, $S_1, S_2, \ldots, S_m$ wherein in set S_1 are placed all vectors $\underline{A}_k$ for which $a_{1k} = 1$; in set S_2 are placed all vectors $\underline{A}_k$ not in S_1 for which $a_{2k} = 1$; and in general, in set S_q are placed all vectors $\underline{A}_k$ not in $S_1, S_2, \ldots,$ or S_{q-1} for which $a_{qk} = 1$, $q \leq m$. When represented in matrix form with column vectors $\underline{A}_1, \underline{A}_2, \ldots, \underline{A}_n$ so ordered, the resulting structure is as shown in Figure 2 where all elements in the unshaded portion of the matrix are zero.

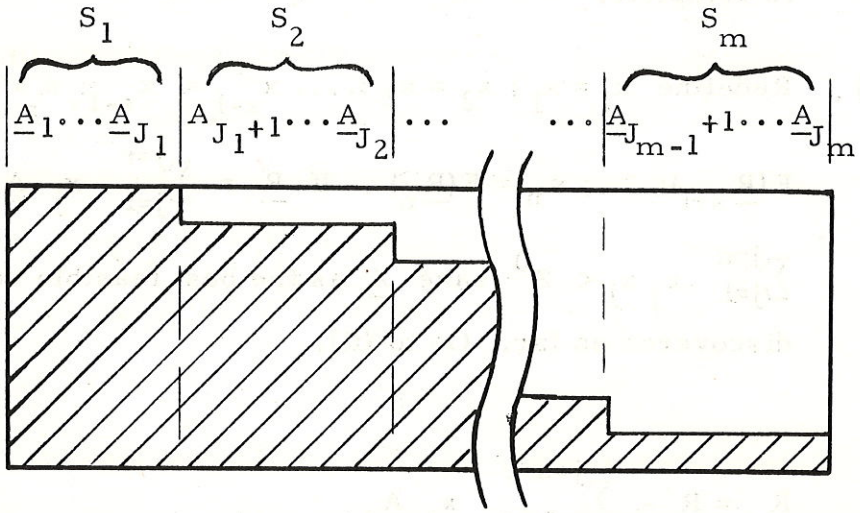

Figure 2. Structural form of matrix of vectors $\underline{A}_1, \underline{A}_2, \ldots, \underline{A}_n$ after preordering by sets. All elements in the unshaded portion of the matrix are zero.

---

[3] As applied to vectors $\underline{A} = (a_1, a_2, \ldots, a_m)$ and $\underline{B} = (b_1, b_2, \ldots, b_m)$ we mean by $\underline{A} \otimes \underline{B} = \underline{C}$ that $c_j = \begin{cases} 1 & \text{if } a_j = b_j = 1 \\ 0 & \text{otherwise} \end{cases}$, $j = 1, 2, \ldots, m$.

[4] A similar pre-ordering of vectors has been employed by Lawler [12] in an algorithm for solving the set covering problem (see [3], p. 286).

Denoting by j_t the number of vectors in S_j, there exists with this ordering no vector $\underline{A}_k$, $k > J_q \equiv \sum_{t=1}^{t=q} j_t$, for which $a_{qk}=1$. Therefore at any stage s there exists no feasible solution to the remaining problem (2) if $r_i^s = 1$ for any i such that $J_i < s$: in such cases the search process can "backtrack" immediately without considering further the variables $x_s, x_{s+1}, \ldots, x_n$.

With this same ordering of vectors a variant of the first feasibility consideration which has considerable "look ahead" power can readily be employed. Suppose stage s has just been completed wherein $\bar{x}_s$ and $\underline{R}_{s+1}$ have been determined. If we denote by t the smallest index i such that $r_i^{s+1} = 1$ then explicit investigation can immediately jump to stage $s' = J_{t-1} + 1$, and resume with $\underline{R}_{s'} = \underline{R}_{s+1}$ since by the ordering of $\underline{A}_k$ we must necessarily have $\bar{x}_k = 0$ for $s < k \leq J_{t-1}$.

In addition to possibilities for reducing search which stem from feasibility considerations there are possibilities based on dominance considerations. If the cost of the best feasible solution $\underline{X}^0$ discovered so far is Z^0 then in subsequent search no explicit consideration need be given potential solutions for which it is known that $Z \geq Z^0$. At stage s, then, investigation may be restricted to potential solutions $\underline{X}$ for which

$$\sum_{j=s}^{j=n} c_j x_j < Z^0 - \sum_{j=1}^{j=s-1} c_j \bar{x}_j \qquad (3)$$

In essence (3) may be viewed as simply another constraint defining a "feasible" solution and the solving of the optimization problem perceived as being accomplished through the solution of a succession of (augmented) feasibility problems.

Denoting by z_s the cost $(Z^0 - \sum_{j=1}^{j=s-1} c_j \bar{x}_j)$ we have first, by analogy with (and in addition to) the first feasibility consideration, that for any particular vector $\underline{A}_s$, $\bar{x}_s = 0$ if $c_s \geq z_s$. Secondly, assuming the ordering of vectors by sets as described above, we have a related dominance consideration having a degree of "look ahead" power as follows. At the completion of stage s suppose t is the smallest index i such that $r_i^{s+1} = 1$. Then there exists no feasible solution to the augmented problem ((2) and (3)) defined at stage $s+1$ if $\phi_t \geq z_{s+1}$, where $\phi_t = \min_{J_{t-1} < j \leq J_t} \{c_j\}$ since (2) requires that $x_j = 1$ for some $j, J_{t-1} < j \leq J_t$. In such cases, the search procedure can backtrack immediately.

The third dominance consideration is one whose implementation requires slightly more problem-solving time but appears to have a fairly high degree of "look ahead" power. Considering again the problem (2) at stage s suppose the constraint equations are summed to form the single constraint $\sum_{j=s}^{j=n} h_j x_j = m_s$, where h_j is the number of unit elements in vector $\underline{A}_j$, $\sum_i a_{ij}$, and m_s is the number of constraints in (1) remaining to be satisfied at stage s, $\sum_{i=1}^{i=m} r_i^s$. Using this single constraint[5] the following subproblem may be formed:

$$\begin{aligned} &\text{Minimize} && \sum_{j=s}^{j=n} c_j y_j = \hat{Z}_s \\ &\text{Subject to:} && \sum_{j=s}^{j=n} h_j y_j = m_s \qquad (4) \\ &\text{and} && y_j = 0 \text{ or } 1,\ j=s,\ s+1,\ldots,n \end{aligned}$$

If Z_s^* denotes the minimum value for (2) then $\hat{Z}_s \leq Z_s^*$ since in (2)

---

[5] In the terminology of Glover [10] this constraint is a "surrogate constraint".

feasibility requires that each of the m individual constraints be satisfied also. Therefore if at any stage s, $\hat{Z}_s \geq z_s$, the search procedure can backtrack immediately since there can exist no solution to (2) resulting in a feasible solution better than $\underline{X}^0$.

The subproblem defined in (4), being one-dimensional knapsack problem, can in practice be solved by any one of a number of algorithms, e.g., by dynamic programming [8], by combinatorial programming [9,14]. In the present context an approach of the latter type might be preferable that has the desirable characteristic that search proceeds first to the discovery of a feasible solution $\underline{Y}$ and then to successively better feasible solutions until ultimately an optimal feasible solution $\underline{Y}^0$ is discovered; with such an algorithm knapsack problem-solving can then be terminated at any time a feasible solution $\underline{Y}$ is discovered for which $\sum_{j=s}^{j=n} c_j y_j < z_s$.

Alternately, rather than actually solve the knapsack problem for $\hat{Z}_s$ we might simply determine a lower bound $\hat{\hat{Z}}_s$ for $\hat{Z}_s$, and check at each stage s whether $\hat{\hat{Z}}_s \geq z_s$. While less powerful, this test requires the expenditure of considerably less problem-solving time to implement. For suppose that vectors $\underline{A}_s, \underline{A}_{s+1}, \ldots, \underline{A}_n$ were ordered so that $c_{(s)}/h_{(s)} \leq c_{(s+1)}/h_{(s+1)} \leq \cdots \leq c_{(n)}/h_{(n)}$. Then by weakening the integer requirements $y_j = 0$ or 1 in (4) to $0 \leq y_j \leq 1$ a lower bound on the cost of satisfying u constraints at stage s is given by the function $\beta_s(r)$ as shown in Figure 3, where $\overline{h}_{(j)} = \sum_{k=1}^{k=j} h_{(k)}$ and $\overline{c}_{(j)} = \sum_{k=1}^{k=j} c_{(j)}$. If, then, at stage s, $\beta_s(m_s) \geq z_s$ the search procedure can backtrack.

Or, an alternative requiring even less implementation time is one in which the quantity $(\delta_s \cdot u)$ is used as a lower bound on $\beta_s(u)$,

where $\delta_s = \min_{j \geq s} \{ c_j / h_j \}$. In this case the test then becomes simply "$\delta_s \cdot m_s \geq z_s$?". This alternative is investigated in the following section.

Finally, there are a number of heuristics that might be employed for ordering the vectors $\underline{A}_k$ within each set S_q as , for instance, "increasing cost, c_j ", or "decreasing number of constraints satisfied, h_j ." For present purposes we employ a heuristic representing a composite of these: vectors will be ordered such that $\underline{A}_j < \underline{A}_k$ in the order implies $c_j / h_j \leq c_k / h_k$.

Imbedding these various considerations and heuristics in the basic enumeration procedure there results an algorithm whose logic is shown in the flow chart of Figure 4. In this procedure record is maintained of only the nonzero elements in the partial solution $\bar{x}_s = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_s)$ defined at each stage s, this being accomplished simply by maintaining

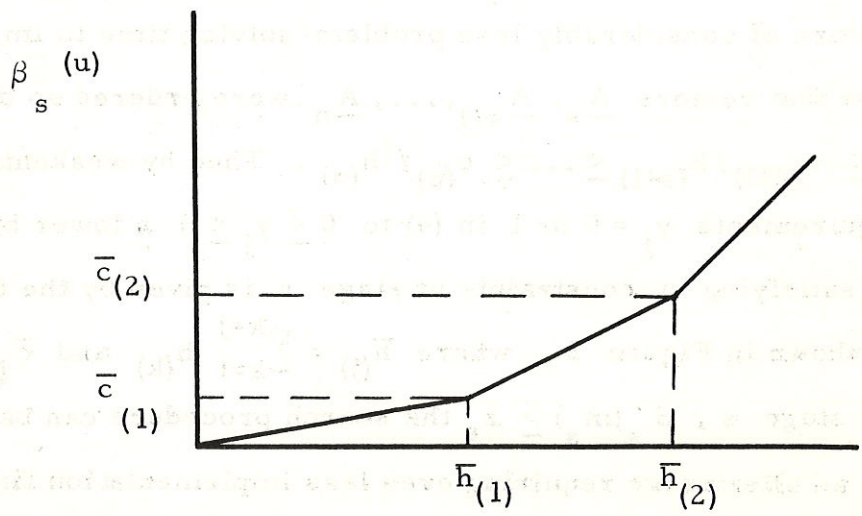

Figure 3. Lower bound $\beta_s(u)$ at stage s on the cost of satisfying u constraints.

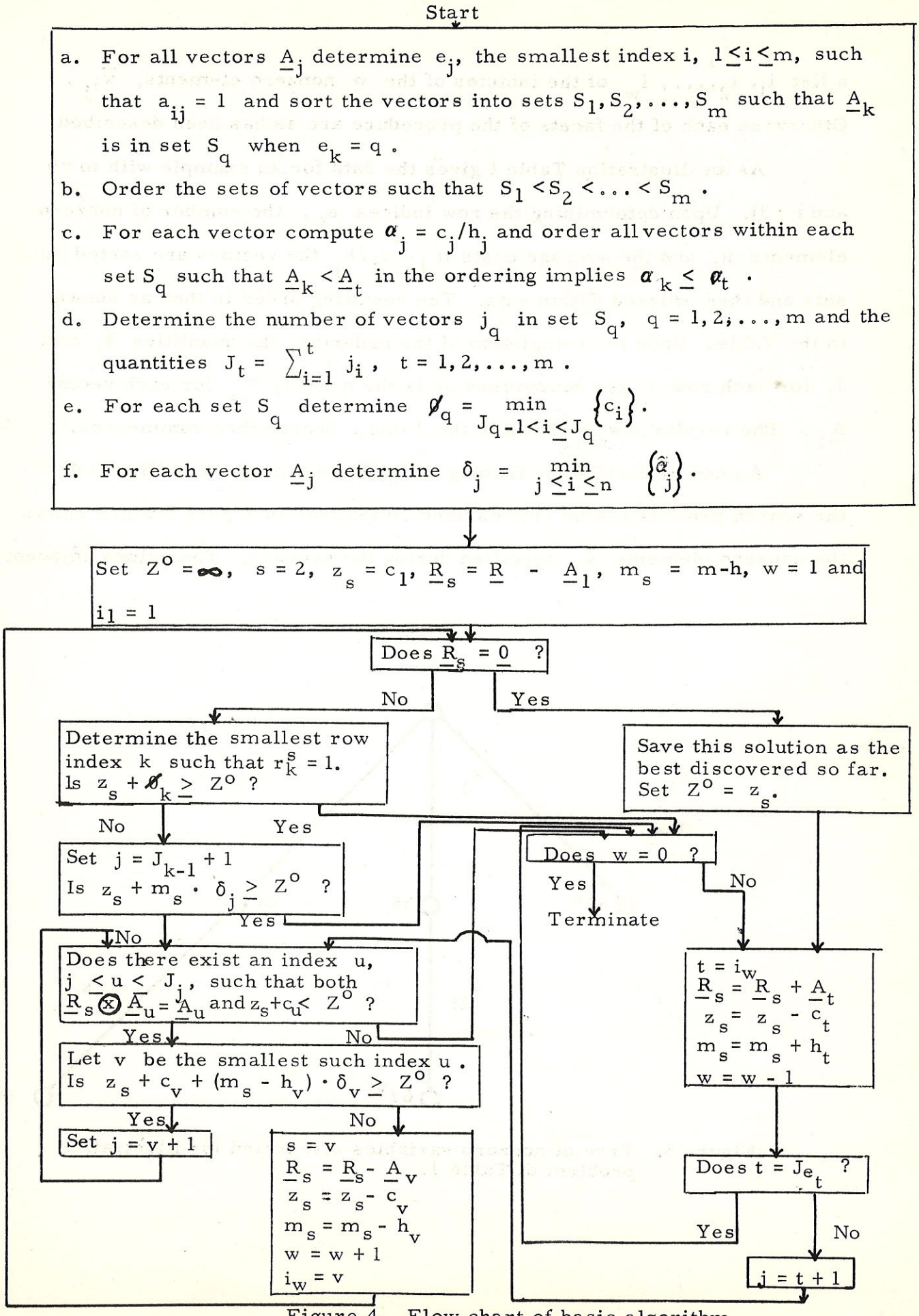

Figure 4. Flow chart of basic algorithm.

a list $i_1, i_2, \ldots, i_w$ of the indicies of the w nonzero elements, $\bar{x}_{i_j}$. Otherwise each of the facets of the procedure are as has been described.

As an illustration Table 1 gives the data for an example with m = 5 and n = 31. Upon determining the row indices e_j, the number of nonzero elements h_j and the average costs $\alpha_j = c_j / h_j$ the vectors are sorted into sets and then ordered within sets. The resulting order is then as shown in the Table. Upon the completion of the ordering, the quantities ϕ_i and J_i for each row i are determined as is the quantity δ_j for each vector $\underline{A}_j$. The results are as shown in the Table. Search then commences.

As can be verified by tracing through the procedure in Figure 4, the search process can be summarized by the tree in Figure 5 which shows the nonzero elements $\bar{x}_j$ specified during the process. The values adjacent

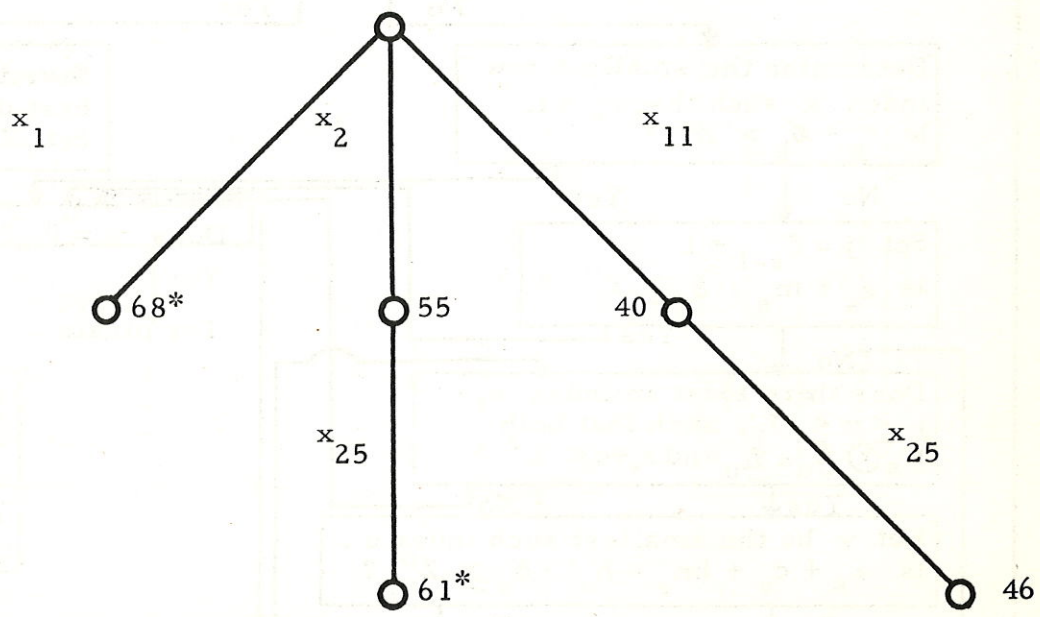

Figure 5. Tree of nonzero variables elaborated for illustrative problem of Table 1.

Table 1. Data for Illustrative All-Zero-One Problem with m=5 and n=31

j	c_j	$\underline{A}_j=(a_{1j}, a_{2j}, \ldots, a_{mj})$					h_j	e_j	α_j	δ_j
1	68	1	1	1	1	1	5	1	13.60	6.00
2	55	1	1	0	1	1	4	1	13.75	6.00
3	57	1	1	1	1	0	4	1	14.25	6.00
4	60	1	0	1	1	1	4	1	15.00	6.00
5	64	1	1	1	0	1	4	1	16.00	6.00
6	49	1	1	0	1	0	3	1	16.33	6.00
7	50	1	1	0	0	1	3	1	16.67	6.00
8	51	1	0	1	0	1	3	1	17.00	6.00
9	54	1	1	1	0	0	3	1	18.00	6.00
10	54	1	0	0	1	1	3	1	18.00	6.00
11	40	1	1	0	0	0	2	1	20.00	6.00
12	62	1	0	1	1	0	3	1	20.67	6.00
13	45	1	0	0	0	1	2	1	22.50	6.00
14	53	1	0	1	0	0	2	1	26.50	6.00
15	58	1	0	0	1	0	2	1	29.00	6.00
16	50	1	0	0	0	0	1	1	50.00	6.00
17	32	0	1	0	1	1	3	2	10.67	6.00
18	45	0	1	1	1	1	4	2	11.25	6.00
19	34	0	1	1	1	0	3	2	11.33	6.00
20	34	0	1	1	0	1	3	2	11.33	6.00
21	23	0	1	1	0	0	2	2	11.50	6.00
22	26	0	1	0	1	0	2	2	13.00	6.00
23	42	0	1	0	0	1	2	2	21.00	6.00
24	31	0	1	0	0	0	1	2	31.00	6.00
25	6	0	0	1	0	0	1	3	6.00	6.00
26	17	0	0	1	0	1	2	3	8.50	8.50
27	47	0	0	1	1	1	3	3	15.67	11.00
28	36	0	0	1	1	0	2	3	18.00	11.00
29	34	0	0	0	1	1	2	4	17.00	11.00
30	28	0	0	0	1	0	1	4	28.00	11.00
31	11	0	0	0	0	1	1	5	11.00	11.00

i	1	2	3	4	5
J_i	16	24	28	30	31
β_i	40	23	6	28	11

to the nodes are the costs incurred up to and including the specification of $\bar{x}_s = 1$; the asterisk indicates that $\underline{R}_s = \underline{0}$ and that a better feasible solution $\underline{X}^0$ has been discovered. First $x_1 = 1$ is specified which results in the first feasible solution with $Z^0 = 68$. The process then backtracks to the origin and proceeds forward, specifying $x_2 = 1$ followed by $x_{25} = 1$. The result is a better feasible solution with $Z^0 = 61$. Backtracking to the previous node ($x_2 = 1$) , the test is made for a vector u , $26 \leq u \leq 28$, such that $\underline{R}_s \otimes \underline{A}_u = \underline{A}_u$. Since no such vector is found the process backtracks again to the origin. The procedure then investigates in turn $x_3 = 1$, $x_4 = 1$,..., $x_{10} = 1$ finding in each case that $\underline{R} \otimes A_u = A_u$ and $c_u < 61$ but that $c_v + (m_v - h_v) \cdot \delta_v \geq 61$. Finally, for $v = 11$ the latter condition fails to hold so that $x_{11} = 1$ is specified, followed by $x_{25} = 1$. Determining the smallest index k such that $r_k^s = 1$ to be 4, and ϕ_4 to be 28, the result is $z_s + \phi_k > 61$. The procedure therefore backtracks, setting $x_{25} = 0$, and proceeds to consider in turn $x_{26} = 1$, $x_{27} = 1$ and $x_{28} = 1$ but in each case $40 + c_v + (3-h_v)\delta_v \geq 61$. Hence, the procedure backtracks again to the origin and proceeds to consider in turn $x_{12} = 1, \ldots, x_{16} = 1$. In each case it is found that $c_v + (5-h_v) \cdot \delta_v \geq 61$. At that point, since $w=0$, problem-solving is complete: the solution $\bar{x}_2 = 1$, $\bar{x}_{25} = 1$ is optimal. Of the 2^{33} nodes in a complete tree of the problem, it was necessary for the procedure to explicitly evaluate but 25.

III. COMPUTATIONAL EXPERIENCE

To investigate the computational feasibility of the general approach computer programs were written in MAD (Michigan Algorithm Decoder) and a number of problems run on an IBM 7094 computer. For discussion purposes we denote as Algorithm 2 the procedure as flow charted in

Figure 4. Algorithm 1 represents the same procedure with the two "knapsack" type tests employing the average costs δ_j and number of remaining unsatisfied constraints m_s omitted.

In Table 2 is shown the solution times[6] for a sample of thirteen problems arising in a truck dispatching context [2, 6, 17]. In this context there is a central depot from which commodities are to be shipped by truck to m destinations. Vector $\underline{A}_j$ in (1) represents the j^{th} truck route, the route starting at the depot and visiting each of the destinations i for which $a_{ij} = 1$. All commodities for destination i are delivered in the same trip. The problem is to select a combination of routes with which to make all m deliveries at minimum cost, the cost of employing route j being c_j.

Each of these sample problems were solved by both Algorithm 1 and Algorithm 2, the solution times in Table 2 representing the time to discover an optimal solution and to prove its optimality. A comparison of times for the two algorithms indicates that the performance of the "knapsack" tests is a good investment of problem-solving time which frequently results in a major reduction in total solution time. For instance, in Example 10 the investment reduces total solution time to less than 1/18 of the problem-solving time required otherwise.

---

[6] The times shown are exclusive of computer input and output time and are accurate to $\pm$ 1/60 of a second.

Table 2. Solution Times for Sample of Truck Dispatching Problems

	Problem Size (m x n)	Solution Times (Seconds) Algorithm 1	Algorithm 2	IPM2	Number of Pivots (IPM 2)
1	5x31	.050	.050	1.867	4
2	6x62	.167	.117	.367	4
3	8x92	.567	.200	2.067	14
4	13x91	35.167	6.367	4.933	33
5	11x231	9.183	1.383	--	--
6	11x561	12.917	2.867	--	--
7	11x1023	27.267	14.383	--	--
8	11x1485	34.950	19.317	--	--
9	12x298	89.133	3.500	--	--
10	12x538	131.033	7.117	--	--
11	12x793	77.667	4.567	--	--
12	15x575	600.000+	69.483	--	--
13	19x1159	---	2400.000+	--	--

+ Problem-solving terminated without having proved optimality

To obtain an indication of the efficiency of these algorithms relative to existing integer programming methods a number of problems were solved by Gomory's all-integer algorithm [11] on the IBM 7094 using SHARE code IPM2 [21] . For those sample problems which were solvable with IPM2 (i.e., those for which $n \leq 100$) the solution time together with the number of pivot steps required is shown in Table 2. From the limited computational experience gained with these algorithms to date the conclusion is that Algorithm 2 is frequently but not invariably preferred to the all-integer algorithm. For instance, in Example 1 Algorithm 2 required less[7] than 1/36 of the time of IPM2, but in Example 4 some 50% more. Further experience is needed to identify attributes of problems which would indicate a priori the preferred method to be used for problems of this size.

As seen from the solution times in the Table, problem-solving time for the combinatorial programming algorithms tends to increase on the average both with the number of vectors n and the number of constraints m in the problem. For a specific problem, however, the time is quite unpredictable. For instance, problem-solving time required for Example 10 exceeds substantially that required for the larger problem of Example 11, where in fact the vectors in the smaller problem are a subset of those in the larger.

As has been mentioned earlier, an important attribute of the combinatorial programming algorithms is that search is directed first to the discovery of a feasible solution and then to successively better and better feasible solutions until ultimately one is discovered that is shown to be optimal. Hence it may be possible to terminate problem-solving with a usable (feasible) solution prior to the ultimate completion of the problem-solving process.

---

[7] This is the problem illustrated in the previous section.

The importance of this attribute can be seen in Example 13 for which the process had not been completed after 40 minutes of computation on the IBM 7094. However, as shown in Table 3 problem-solving could have been terminated any time after the first 5.6 seconds with a feasible solution. In Table 3 is shown the rate of problem-solving progress for this example together with two others. For each example the Table shows the value of the objective function Z for each successive feasible solution discovered together with the elapsed time t at which it was discovered.[8] The first elapsed time for each example includes both the time necessary to arrange the vectors from an arbitrary input order into the prescribed order for search and the subsequent time required to find a first feasible solution. As these results indicate the time required to find the first feasible solution tends in general to constitute a rather small fraction of the total problem-solving time. In addition, they suggest[9] that a large fraction of the total time is commonly expended in proving optimality, a characteristic that has also been observed with other combinatorial programs.[10]

While most of the problems investigated to date have been of the truck dispatching type, a number of random problems have been solved with the algorithms. The resulting times for 6 problems with n=60 and 100, and m=10, 20 and 35 are shown in Table 4. In each case the problem was formed by generating (n-m) vectors $\underline{A}_j$ with h_j randomly distributed nonzero elements, h_j being uniformly distributed over the range $1 \leq h_j \leq m$, and affixing to these (n-m) vectors the m units vectors. For all n vectors the cost c_j was a uniformly distributed random variable in the range $0 \leq c_j \leq 100$.

8 The difference in total time between Tables 1 and 2 for Examples 4 and 10 is the time required to print intermediate solutions in the latter case.

9 In Example 13 it is conjectured that Z = 683 is an optimal value.

10 See, for instance, the experience reported in [15] with a class of sequencing problems.

EXAMPLE 5		EXAMPLE 10		EXAMPLE 13	
Objective Function Value Z	Elapsed Problem-Solving Time t	Objective Function Value Z	Elapsed Problem-Solving Time t	Objective Function Value Z	Elapsed Problem-Solving Time t
891	.183	342	2.333	814	5.617
853	.200	326	2.350	784	5.633
823	.267	322	2.383	780	5.683
821	.300	310	2.400	733	5.717
818	1.417	308	2.517	728	59.867
814	1.517	306	2.550	709	60.133
811	1.567	302	2.583	707	132.417
785	1.650	290	2.700	706	132.867
784	1.683	Optimality Proved	7.367	702	156.300
778	1.750			697	265.483
771	3.583			692	265.650
770	4.067			683	349.767
Optimality Proved	6.617			Non-Optimality Termination	2400.000+

Table 3. Problem-Solving Time For Successive Feasible Solutions

Table 4. Solution Times for Sample of Randomly Generated Problems

	Problem Size (m x n)	Solution Times (Seconds) Algorithm 1	Algorithm 2
a	10x60	.117	.150
b	10x100	.250	.283
c	20x60	.067	.167
d	20x100	.167	.283
e	35x60	.217	.217
f	35x100	.217	.283

In contrast to the results for the truck dispatching type problems shown in Table 2, these results indicate that investment of problem-solving time in performing "knapsack" tests is a poor investment: the added time spent in implementing the tests is not offset by a commensurate reduction in search time. Of the two algorithms, Algorithm 1 is thus preferred for the randomly generated problems. This difference in the effectiveness of the algorithms when applied to the two type of problems stems from differences in the characteristics of the problems. The truck dispatching problems are characterized by the facts that there tend to be a large number of feasible solutions to the problem and a high correlation between the h_j and the c_j with the consequence that the lower bounds used in the tests tend to be close to the minimum attainable cost for a feasible solution; hence the discrimination power of the tests tends to be sufficiently strong to effectively reduce search. On the other hand, for randomly generated problems the lower bounds tend to markedly understate the attainable cost for feasible solutions, and hence to be less effective in pruning the tree. In the extreme case there is no resultant pruning, and total problem-solving time is simply extended by the time expended on fruitless testing. Thus for randomly generated problems this time should be eliminated,

or perhaps more effectively expended in performing additional feasibility tests, some of which are suggested in the following section.

IV. MODIFICATIONS OF THE BASIC ALGORITHM

While the computational feasibility of the general approach for solving all-zero-one type problems has been demonstrated with these basic algorithms, there are a number of simple modifications to these procedures which might result in more efficient procedures.

For instance, Balinski [3] has noted that there are a number of pre-analysis considerations that might profitably be applied to the problem prior to the actual execution of the algorithms. First, in the absence of knowledge regarding the structure of the vectors $\underline{A}_j$ comprising the problem, it would most likely be worthwhile to ascertain that for each row i, $a_{ij} = 1$ for at least one j, for otherwise there exists no feasible solution to the problem. Secondly, it might be possible to reduce the number of vectors n in the problem through dominance considerations: vector $\underline{A}_k$ may be eliminated from the problem if there exists vectors $\underline{A}_{j_i}$ such that $\sum_i \underline{A}_{j_i} = \underline{A}_k$ and $\sum_i c_{j_i} \leq c_k$. Thirdly, should there exist a row i with only a single vector $\underline{A}_k$ having a nonzero element a_{ik}, this row and column vector may be removed, yielding a smaller problem of dimensions $(m-1) \times (n-1)$ with requirements $\underline{R}' = \underline{R} - \underline{A}_k$. Finally, if by permuting rows and columns of the matrix T formed by the column vectors $\underline{A}_j$ it is possible to form a matrix T' having the diagonal structure as shown in Figure 6, then an optimal solution can be obtained by decomposing the problem into the p subproblems defined by the diagonal submatrices T_i' and solving each of them independently.

Within the algorithms themselves, more efficient procedures may result through use of additional feasibility and/or dominance considerations.

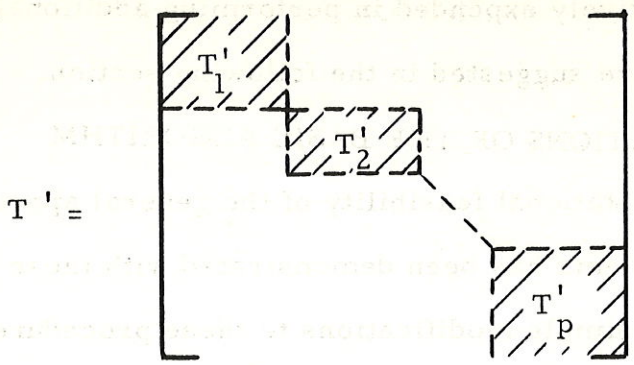

Figure 6. Structural form of decomposable problem. All elements in the unshaded portion of the matrix are zero.

For instance, it may be advantageous at each stage s to verify that the feasibility test $\gamma_t \leq m_s$ is satisfied, where m_s is the number of constraints to be satisfied at stage s and $\gamma_t = \min_{s \leq u \leq J_t} \{h_u\}$ for the smallest row index t for which $r_i^s = 1$; whenever the test is not satisfied the search process can backtrack immediately. Another feasibility consideration of potential value within the algorithm itself, especially when solving random problems, is that described earlier as a pre-analysis consideration wherein there must be at least one nonzero element a_{ij} for each i. Similarly, at each stage s during problem-solving there must exist for each i such that $r_i^s = 1$ at least one vector $\underline{A}_j$, $s \leq j \leq n$, such that $a_{ij} = 1$. This test can be easily implemented by defining recursively, after ordering the vectors at the outset of the algorithm, the vectors $\underline{E}_j$, $j = n, n-1, \ldots, 1$, where[11] $\underline{E}_j = \underline{E}_{j+1} \oplus \underline{A}_j$ and $\underline{E}_n = \underline{A}_n$; and then, throughout problem-solving, verifying at each stage s that $\underline{R}_s \otimes \underline{E}_s = \underline{R}_s$.

With regard to dominance considerations, as was suggested in Section II a test more stringent than $\delta_s \cdot m_s < z_s$ can be devised by using additional segments in the function $\beta_s(u)$ shown in Figure 2 to get a higher lower bound $\hat{Z}_s$ on the remaining cost Z_s^*. An alternative to this which would

[11] The symbol $\oplus$ denotes the logical "or" operation; as applied to vectors $\underline{A} = (a_1, a_2, \ldots, a_m)$ and $\underline{B} = (b_1, b_2, \ldots, b_m)$ we mean by $\underline{A} \oplus \underline{B} = \underline{C}$ that $c_j = \begin{cases} 0 \text{ if } a_j = b_j = 0 \\ 1 \text{ otherwise} \end{cases}$ for $j = 1, 2, \ldots, m$.

require considerably more setup effort, storage, and implementation time but which would be expected to yield higher lower bounds on Z_s^* is to employ the bound $\hat{Z}_s = \sum_{i=1}^{i=m} r_i^s \not{\theta}_{it}$, where $r_i^s = 0$ for $i < t$,

$$\not{\theta}_{it} = \begin{cases} \not{\theta}_{i,t+1} & \text{for } a_{it} = 0 \\ \min \left[\not{\theta}_{i,t+1}, c_t / h_t \right] & \text{otherwise} \end{cases}, \quad t=1,2,\ldots,n-1$$

and

$$\not{\theta}_{in} = \begin{cases} \infty & \text{for } a_{in} = 0 \\ c_n / h_n & \text{otherwise} \end{cases}$$

This test would subsume the last feasibility consideration given in the preceeding paragraph. Or, possibly it would prove more efficient to store and base tests only on the values $\not{\theta}_{it}$ for $t = 1, J_1 + 1, J_2 + 1, \ldots, J_{m-1} + 1$.

With the exception of the first and third pre-analysis considerations, the problem-solving time that would be consumed in implementing these modifications would not appear to be insignificant. Further investigation is required to determine whether, on the average, the reduction in search time occasionned by the use of these added considerations exceeds the time spent in implementing them.

Finally, there are two general possibilities for improvement which are applicable to all combinatorial programming procedures employing a strategy of the type used in the present algorithms: pre-specification of upper and/or lower bounds on the value of the objective function, $Z^U \geq Z \geq Z^L$,

and searching by "successive approximations". Clearly if values of Z^U and/or Z^L can be pre-specified to the algorithm at the outset, the possibility exists of reducing the amount of search time required by the algorithm - perhaps quite dramatically, as in cases when the values are close to or equal to the optimum value. In some instances such values may be readily supplied from previous experience, related problems, etc., without the necessity of expending significant amounts of problem-solving effort. More generally, this possibility for overall improvement through pre-specification of bounds leads to the general quest for "good" ways to get "good" initial solutions and bounds, and to the question of the optimal allocation of total problem-solving time between the preliminary search for "good" bounds to pre-specify to the algorithm and the execution of the algorithm per se. In the following section a strategy in which bounds are determined by linear programming is described, but in general the topic is beyond the scope of the present paper.

In searching by "successive approximations" the basic algorithm is modified so that in searching for successively better and better feasible solutions it is now required that each feasible solution exhibit an improvement over its predecessor of at least an amount[12] ∇. By imposing this requirement it is hoped that greater portions of the tree will be pruned and problem-solving time thereby reduced. Upon termination a solution will have been discovered with a solution value Z^* that is within ∇ of the optimum: $Z^* \geq Z > Z^* - \nabla$. At this point the algorithm can be re-applied, if desired, to search the intermediate range[13] for an optimal solution, or

---

[12] In the basic algorithm this modification consists simply in substituting the quantity $(Z^o - \nabla)$ for Z^o in (3).

[13] This strategy thus entails the re-elaboration and re-evaluation (at least implicitly) of portions of the tree that have already been considered earlier. With the enumeration procedures employed in the algorithms in this paper the reconsideration applies to paths in the tree corresponding to potential solutions lexicographically smaller than the solution which yielded the value Z^*.

for one within, say β of an optimum.

Both of these general possibilities for improvement remain topics for future investigation.

V. EXTENSION TO THE ALL-ZERO-ONE PROBLEM WITH INEQUALITIES

With minor changes the discussion and algorithms for the all-zero-one problem of form (1) can be extended to the problem with inequalities of the form given by (1'). In this last section we comment briefly on this extension.

Paralleling the discussion in Section II, we will assume in the basic enumeration process for problems of form (2) that the column vectors $\underline{A}_j$ are pre-ordered in a fixed order, and that potential solutions $\underline{X}=(x_1, x_2, \ldots, x_n)$ are generated in lexicographically decreasing order. Corresponding to (2) we have that at stage s-1 in the enumerating process, having specified values $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{s-1}$, there remains a problem of the form (2'):

$$\text{Minimize} \quad \sum_{j=s}^{j=n} c_j x_j$$

$$\text{Subject to:} \quad \sum_{j=s}^{j=n} x_j \underline{A}_j \geq \underline{R} - \sum_{j=1}^{j=s-1} \bar{x}_j \underline{A}_j \equiv \underline{R}_s \qquad (2')$$

$$\text{and} \quad x_j = 0 \text{ or } 1 \qquad j=s, s+1, \ldots, n$$

From the nonnegativity of the elements a_{ij} and the inequality in (2) it follows that for each k, $1 \geq r_k^1 \geq r_k^2 \geq \ldots \geq r_k^s \geq \ldots \geq r_k^n > -\infty$. The k^{th} inequality becomes satisfied as soon as a stage t is reached for which $r_k^t \leq 0$; thereafter in the formulation of problem (2') at stage s, $s > t$ it is equivalent to set $r_k^s = 0$. That is, in general we set $r_k^s = \max \left[0, r_k - \sum_{j=1}^{s-1} a_{kj} \bar{x}_j\right]$. The importance of this equivalence is that we can thus continue to represent the elements of vectors $\underline{R}_s$ as bits in binary words. Computationally we now have for $\underline{R}_s$:

$$\underline{R}_s = \underline{R}_{s-1} - \underline{R}_{s-1} \otimes (\bar{x}_{s-1} \cdot \underline{A}_{s-1}) = \underline{R}_{s-1} \otimes (\bar{x}_{s-1} \cdot \underline{A}_{s-1}), \quad s=2, 3, \ldots, n$$

where $\overline{\underline{A}}_{s-1}$ denotes the column vector whose elements are the binary complements of the elements of $\underline{A}_{s-1}$.

With this equivalence the basic ennumeration procedure for problems of form (2) becomes:

(i) Set $x_1 = F(\underline{R}_1)$; $x_2 = F(\underline{R}_2); \ldots; x_n = F(\underline{R}_n)$.

If $\underline{R}_n = \underline{0}$ save $\underline{X}$ as the best feasibly solution $\underline{X}^o$ discovered so far, setting $Z^o = \sum_{j=1}^{j=n} c_j x_j^o$. Go to (ii).

(ii) If there is a j, $1 \leq j < n$ such that $x_j = 1$ let s be the largest such j and go to (iii). Otherwise problem-solving is complete.

(iii) Redefine $x_1 = x_1$; $x_2 = x_2; \ldots; x_{s-1} = x_{s-1}$; $x_s = 0$; $x_{s+1} = F(\underline{R}_{s+1})$; $\ldots; x_n = F(\underline{R}_n)$. If $\underline{R}_n = \underline{0}$ and $\sum_{j=1}^{j=n} c_j x_j < Z^o$ save $\underline{X}$ as the best solution discovered so far. Go to (ii).

where $\underline{0}$ is the m-dimensional column vector with all elements zero and[14]

$$F(\underline{R}_s) = \begin{cases} 0 \text{ if } \underline{R}_s \otimes \underline{A}_s = \underline{0} \\ 1 \text{ otherwise} \end{cases}$$

Continuing on, it follows as in the case of problem (2) that a feasible solution can exist to problem (2') only if there exists for each k such that $r_k^s = 1$ at least one vector $\underline{A}_t$ with $a_{kt} = 1$, $t \geq s$. This consideration can be simply implemented in a manner similar to that used for problem (2) by appropriately defining sets $S_1', S_2', \ldots, S_m'$ of the vectors $\underline{A}_k$ and pre-ordering vectors on the basis of these sets. In the present case, however, this is accomplished at the expense of a marked increase in problem size[15] resulting from the replication of vectors $\underline{A}_k$. For the present problem

14 That $\overline{x}_s = 0$ when $\underline{R}_s \otimes \underline{A}_s = \underline{0}$ follows from dominance considerations, and that $\overline{x}_s = 1$ when $\underline{R}_s \otimes \underline{A}_s \neq \underline{0}$ from feasibility considerations.

15 Alternately it can be accomplished at the expense of additonal bookkeeping which is probably more efficient computationally, especially when $m > B$. The present discussion is facilitated, however, by assuming the vectors explicitly replicated.

sets $S'_1, S'_2, \ldots, S'_m$ are defined such that S'_j contains all vectors for which $a_{jk} = 1$. If vector $\underline{A}_k$ is then replicated $\left(\sum_{j=1}^{j=m} a_{jk}\right)$ times and placed in each of the relevant sets, and the $N = \sum_{jk} a_{jk}$ vectors pre-ordered so that those in set S'_1 precede those in set S'_2, etc., we can then proceed exactly as in case of problem (1). That is, if j_t is the number of vectors in S'_t, and $J_q = \sum_{t=1}^{t=q} j_t$, then the search process after specifying $\bar{x}_s$, $s=J_i$, $i=1,2,\ldots,m-1$, can backtrack if $r_i^s = 1$. Also, in the forward search process if t denotes the smallest index i such that $r_i^{s+1}=1$ at the completion of stage s, then explicit investigation can jump to stage $s'=J_{t-1}+1$ and resume with $\underline{R}_{s'}=\underline{R}_{s+1}$.

Proceeding to dominance considerations we have as before: (i) $\bar{x}_s = 0$ if $c_s \geq z_s$; (ii) search may backtrack at stage s+1 if $\not{\theta}_t \geq z_{s+1}$, where $\not{\theta}_t = \min_{J_{t-1}<j\leq J_t} \{c_j\}$ and t is the smallest index i such that $r^{s+1} = 1$; and (iii) search may backtrack at stage s if $\hat{Z}_s \geq Z_s^*$ or $\hat{\hat{Z}} \geq Z_s^*$, where $\hat{Z}_s$ is a lower bound on the value of an optimal solution Z_s to the knapsack problem defined by (4) with the constraint $\sum_{j=s}^{j=N} h_j y_j = m_s$ replaced by the inequality $\sum_{j=s}^{j=N} h_j y_j \geq m_s$. In this latter instance all of the bounding considerations discussed in Section II apply to the present problem. In the present case the quantity h_j for vector $\underline{A}_j$ is redefined to be $h_j = \sum_{i=q}^{i=m} a_{ij}$ where $\underline{A}_j$ is a member of set S'_q.

Finally, turning to the modifications in Section IV each of Balinski's four pre-analysis considerations apply when the second is appropriately changed for the present problem: vector $\underline{A}_k$ may be removed from the problem if there exists vectors $\underline{A}_{j_i}$ such that $\sum_i \underline{A}_{j_i} \geq \underline{A}_k$ and $\sum_i c_{j_i} \leq c_k$. Within the algorithm itself the test for $\hat{Z}_s \geq Z_s^*$ remains valid, where $Z_s = \sum_i r_i^s \not{\theta}_{it}$. (Again the quantity h_t appearing in the expressions for $\not{\theta}_{it}$ is redefined, as above.) And, of course, both of the general possibilities for improvement apply to the present problem: pre-specification of upper and/or lower bounds on the objective function, and searching by "successive approximations."

As an illustration we consider again the problem with m=5 and n=31 in Table 1. Applying Balinski's second pre-analysis dominance consideration twenty of the original vectors $\underline{A}_j$ are eliminated, leaving vectors j = 2, 6, 7, 10, 11, 13, 17, 21, 22, 25 and 31. Upon replicating each of these remaining vectors $\underline{A}_j$ to form the sets $S_1', S_2', \ldots, S_5'$ there results $\sum_{i,j} a_{ij} = 26$ vectors $\underline{A}_j$. Sorting these vectors into sets, and ordering within sets in increasing value of $\alpha_j = c_j/h_j$, there results the equivalent problem whose data is given in Table 5. Proceeding with the solution of the problem, the result is as indicated by the tree in Figure 7 which shows the non-zero variables specified during the process. First $x_1 = 1$ is specified, followed by $x_{14} = 1$ to yield a feasible solution with Z=61. The process then backtracks to the node at $x_1 = 1$ and considers in turn j=15 and 16 but finds in both cases $z_s + c_j > 61$. Hence the process backtracks to the origin and investigates in turn j=2, 3 and 4 finding that $c_j < 61$ but that $c_j + (m - h_j) \cdot \delta_j \geq 61$. For j=5 $c_j + (m-h_j) \cdot \delta_j < 61$ so that we specify $x_5 = 1$ followed by $x_{14} = 1$. Then, however, $z_s + \alpha_4 = 46 + 26 > 61$ so the process backtracks to the node $x_5 = 1$ and tries j=15. Since $z_5 + c_{15} = 63 > 61$ it backtracks again to the origin. For j=6, $c_6 + (m-h_6)\,\delta_6 = 45 + 3(6) > 61$ and, since w=o, problem-solving is complete.

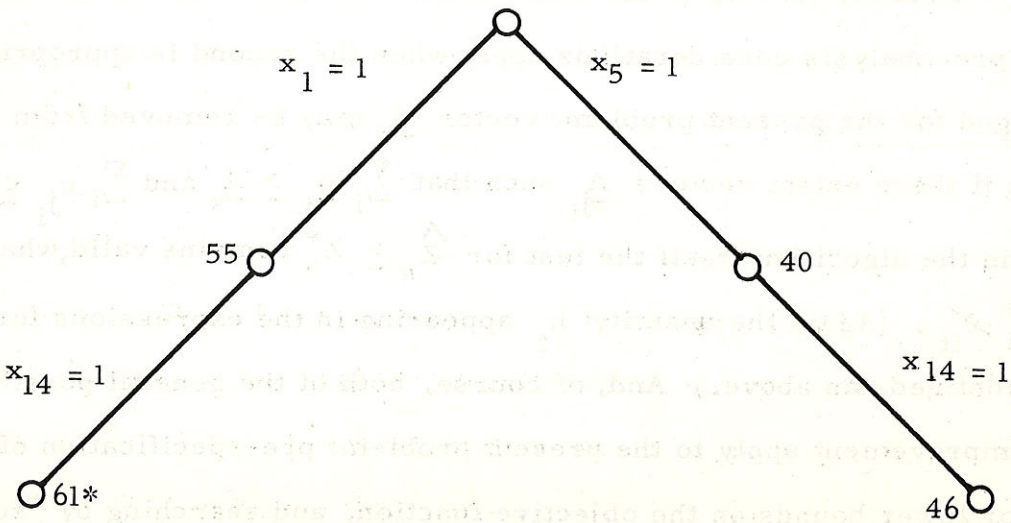

Figure 7. Tree of nonzero variables elaborated for illustrative problem of Table 5.

Table 5. Data for Illustrative All-Zero-One Problem With Inequalities as Derived from Problem in Table 1

j	$\bar{j}$	c_j	$A_j=(a_{1j}, a_{2j}, \ldots, a_{mj})$	h_j	e_j	α_j	δ_j
1	2	55	1 1 0 1 1	4	1	13.75	6.00
2	6	49	1 1 0 1 0	3	1	16.33	6.00
3	7	50	1 1 0 0 1	3	1	16.67	6.00
4	10	54	1 0 0 1 1	3	1	18.00	6.00
5	11	40	1 1 0 0 0	2	1	20.00	6.00
6	13	45	1 0 0 0 1	2	1	22.50	6.00
7	17	32	0 1 0 1 1	3	2	10.67	6.00
8	21	23	0 1 1 0 0	2	2	11.50	6.00
9	22	26	0 1 0 1 0	2	2	13.00	6.00
10	2	55	1 1 0 1 1	3	2	18.33	6.00
11	6	49	1 1 0 1 0	2	2	24.50	6.00
12	7	50	1 1 0 0 1	2	2	25.00	6.00
13	11	40	1 1 0 0 0	1	2	40.00	6.00
14	25	6	0 0 1 0 0	1	3	6.00	6.00
15	21	23	0 1 1 0 0	1	3	23.00	11.00
16	17	32	0 1 0 1 1	2	4	16.00	11.00
17	22	26	0 1 0 1 0	1	4	26.00	11.00
18	10	54	1 0 0 1 1	2	4	27.00	11.00
19	2	55	1 1 0 1 1	2	4	27.50	11.00
20	6	49	1 1 0 1 0	1	4	49.00	11.00
21	31	11	0 0 0 0 1	1	5	11.00	11.00
22	17	32	0 1 0 1 1	1	5	32.00	32.00
23	13	45	1 0 0 0 1	1	5	45.00	45.00
24	7	50	1 1 0 0 1	1	5	50.00	50.00
25	10	54	1 0 0 1 1	1	5	54.00	54.00
26	2	55	1 1 0 1 1	1	5	55.00	55.00

i	1	2	3	4	5
J_i	6	13	15	20	26
δ_i	40	23	6	26	11

In conclusion, we comment briefly on possibilities for improvement through pre-specification of bounds. As remarked in the previous section it may in practice be more efficient to first obtain by some means upper and lower bounds for the optimal value of Z, Z^L and Z^U, and then employ the combinatorial programming procedure for reliably searching the intermediate range. For the present problem both bounds can be readily derived from an optimal noninteger solution to the problem given in (2) as determined by a linear programming algorithm. If $\underline{Y} = (y_1, y_2, \ldots, y_n)$ represents an optimal noninteger solution to (2) then[16] $Z^L = \sum_{j=1}^{j=n} c_j y_j$ and $Z^U = \sum_{j=1}^{j=n} c_j \overline{y}_j$ where $\overline{y}_j = \langle y_j + .99\ldots9 \rangle$, the representation $\langle \alpha \rangle$ denoting the largest integer contained in $\langle \alpha \rangle$.

As an alternative, problem-solving time might first be spent in searching for a rounded feasible integer solution $W = (w_1, w_2, \ldots, w_n)$ to (2) with value $\sum_{j=1}^{n} c_j w_j$ closer to optimum than $\overline{Y} = (\overline{y}_1, \overline{y}_2, \ldots, \overline{y}_n)$, thereby reducing the range $Z^L \leq Z < Z^U$ to be searched. With reference again to the optimal noninteger solution Y, the solution $\overline{W} = (\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_n)$ derived by setting $\overline{w}_j = \langle y_j \rangle$ for all j constitutes an optimal integer solution to the problem with requirements $\sum_{j=1}^{j=n} \overline{w}_j \underline{A}_j$. With these requirements fulfilled by $\overline{W}$ there remains a smaller problem of the form of (2) with requirements $\underline{R} - \sum_{j=i}^{j=n} \overline{w}_j \underline{A}_j$. If $\overline{\overline{W}} = (\overline{\overline{w}}_1, \overline{\overline{w}}_2, \ldots, \overline{\overline{w}}_n)$ represents any feasible solution to this remaining problem then $W = \overline{W} + \overline{\overline{W}}$ constitutes a feasible solution to the original problem having requirements $\underline{R}$. If $Z^L - \sum_{j=1}^{j=n} c_j \overline{w}_j \leq \sum_{j=1}^{j=n} c_j \overline{\overline{w}}_j < Z^U - \sum_{j=1}^{j=n} c_j \overline{w}_j$ solution W constitutes a feasible solution with value Z closer to optimal than $\overline{Y}$; in the special event that $\sum_{j=1}^{j=n} c_j \overline{\overline{w}}_j = Z^L - \sum_{j=1}^{j=n} c_j \overline{w}_j$ solution W constitutes an optimal solution.

---

16 In the event that all coefficients c_j are integers, the higher lower bound $\left\langle \sum_{j=1}^{n} c_j y_j + .99..9 \right\rangle$ may be used.

Pursuing this latter alternative, a reasonable problem-solving strategy would appear to be to first obtain an optimal noninteger solution Y for the original problem. Then from Y formulate the remaining subproblem with requirements $\underline{R} - \sum_{j=1}^{j=n} \bar{w}_j \underline{A}_j$ with bounds $Z^L - \sum_{j=1}^{j=n} c_j \bar{w}_j$ and $Z^U - \sum_{j=1}^{j=n} c_j \bar{w}_j$, and search by the combinatorial programming procedure the intermediate range, obtaining an optimal integer subproblem solution $\bar{\bar{W}}^*$. Forming $W = \bar{W} + \bar{\bar{W}}^*$ we test if $\sum_{j=1}^{j=n} c_j w_j = Z^L$; if so, problem-solving is complete; otherwise the combinatorial programming procedure is applied to the original problem to investigate the range $Z^L \leq Z < \sum_{j=1}^{j=n} c_j w_j$. In the context of the basic one-dimensional cutting stock problem [16] this general strategy was found to be quite successful in that for each of the approximately one hundred cases investigated, solution W constituted an optimal solution, making it unnecessary to subsequently apply the combinatorial programming procedure to the original problem. For the present class of all-zero-one problems, possibly the investigations of Balinski and Norman (see [4]) will yield information regarding the effectiveness of problem-solving strategies of this type.

REFERENCES

1. Balas, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables" Operations Research, Vol. 13, No. 4 (July-August, 1965) p. 517-545.

2. Balinski, M. L. and Quandt, R. E., "On an Integer Program for a Delivery Problem", Operations Research, Vol. 12, No. 2 (March-April, 1964), p. 300-304.

3. Balinski, M. L., "Integer Programming: Methods, Uses, Computation" Management Science, Vol. 12, No. 3 (November, 1965)p. 253-313.

4. Balinski, M. L., "Integer Programming" (Abstract), Chapter 6 of Pierce, J. F., (Ed), Operations Research and the Design of Management Information Systems, Technical Association of Pulp and Paper Industry, New York, N. Y., 1967.

5. Crowston, W. C., "Network Planning Models", forthcoming.

6. Dantzig, G. B. and Ramser, J. H., "The Truck Dispatching Problem", Management Science, Vol. 6, No. 1 (October, 1959)p. 80-91.

7. Day, R. H., "On Optimal Extracting From a Multiple File Data Storage System: An Application of Integer Programming", Operations Research, Vol. 13, No. 3 (May-June, 1965)p. 482-494.

8. Gilmore, P. C. and Gomory, R. E., "A Linear Programming Approach to the Cutting Stock Problem", Operations Research, Vol. 9, No. 6 (November-December, 1961) p. 849-859.

9. Gilmore, P. C. and Gomory, R. E., "A Linear Programming Approach to the Cutting Stock Problem-Part II", Operations Research, Vol. 11, No. 6 (November-December, 1963)p. 863-888.

10. Glover, F., "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem", Operations Research, Vol. 13, No. 6 (November-December, 1965) p. 879-919.

11. Gomory, R. E., "An All-Integer Integer Programming Algorithm", Chapter 13 of Muth, J. F. and Thompson, G. L. (Editors), Industrial Scheduling, Prentice-Hall, Englewood Cliffs, New Jersey, 1963.

12. Lawler, E. L., "An Algorithm for Solving Covering Problems", University of Michigan, mimeograph report, December, 1964.

13. Little, J. D. C., Murty, K. G., Sweeney, D. W. and Karel, C., "An Algorithm for the Traveling Salesman Problem", Operations Research, Vol. 11, No. 6 (November-December, 1963) p. 972-989.

14. Pierce, J. F., Some Large Scale Production Scheduling Problems in the Paper Industry , Prentice-Hall, Englewood Cliff, New Jersey, 1964.

15. Pierce, J. F. and Hatfield, D. J., "Production Sequencing by Combinatorial Programming", Chapter 17 of Pierce, J. F., (Ed), Operations Research and the Design of Management Information Systems, Technical Association of Pulp and Paper Industry New York, N. Y., 1967.

16. Pierce, J. F., "On the Solution of Integer Cutting Stock Problems by Combinatorial Programming - Part I", IBM Cambridge Scientific Center Technical Report 36. Y02, May, 1966.

17. Pierce, J. F., "On the Truck Dispatching Problem", IBM Cambridge Scientific Center Technical Report 36. Y07, forthcoming.

18. Root, J. G., "An Application of Symbolic Logic to a Selection Problem", Operations Research, Vol. 12, No. 4 (July-August, 1964) p. 519-526.

19. Rossman, M. J. and Twery, R. J., "Combinatorial Programming". Unpublished paper presented at 6th Annual Meeting of the Operations Research Society of America, Boston, Massachusetts, 1958.

20. Salveson, M. E., "The Assembly Line Balancing Problem", Journal of Industrial Engineering, Vol. VI, No. 3 (May-June, 1955) p. 18-25.

21. Wade, C. S. and Gomory, R. E., IPM 2, Share Distribution Number 1191, September 1961.

IBM