Discussion Paper No. 8

AN ALGORITHM FOR LARGE SET PARTITIONING PROBLEMS

by

Roy E. Marsten

July 14, 1972

ABSTRACT

An algorithm is presented for the special integer linear program known as the set partitioning problem. This problem has a binary coefficient matrix, binary variables, and unit resources. Furthermore, all of its constraints are equations. In spite of its very special form, the set partitioning problem has many practical interpretations. The algorithm is of the implicit enumeration type. A special class of finite mappings is enumerated rather than the customary set of binary solution vectors. Linear programming is used to obtain bounds on the minimal costs of the subproblems that airse. Computational results are reported for several large problems.

1. Introduction

The integer linear program of the special form:

$$\text{(PP)} \quad \text{minimize} \quad \sum_{j=1}^{n} c_j y_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} y_j = 1 \quad \text{for } i=1,\ldots,m$$

$$y_j = 0 \text{ or } 1 \quad \text{for } j=1,\ldots,n$$

$$\text{where } a_{ij} = 0 \text{ or } 1 \text{ for all } i, j \text{ and } c_j \geq 0 \text{ for all } j,$$

is known as the set partitioning problem. It is a special case of the well-known set covering problem which has inequality ($\geq$) rather than equality constraints. The name "set partitioning problem" comes from the following interpretation. Think of each column of $A = (a_{ij})$ as a subset A_j of the index set $I = \{1,\ldots,i,\ldots,m\}$ where $i \in A_j$ if and only if $a_{ij} = 1$. The problem is then to select a set of columns which gives a minimal cost partition of I.

1.1 Applications

Covering and partitioning problems have been studied widely because of their many practical applications and because of their intriguing binary structure. Applications for the partitioning problem which have appeared in the literature include airline crew scheduling [1,23,38,40], airline fleet scheduling [26], truck routing [4,6,10,24,30], political districting [16,43], information retrieval [12], symbolic logic [8], switching theory [3,7,29,34], stock cutting [31], line balancing [35], capacity balancing [39], PERT-CPM [8], capital investment [42], coloring problems [5], location of offshore drilling platforms [9], and facilities location [33].

Many of these references are discussed in the recent survey by Garfinkel [14]. The airline crew scheduling problem will be described here, since this application was the main impetus to the development of the present algorithm and the source for the numerical data used to test it.

Several major airlines employ a model of the crew scheduling process that first appeared in [38]. This model can be described as follows. Given the airline's timetable, a set of possible crew rotations can be generated. Each crew rotation is a sequence of scheduled flight segments constituting a round trip - that is, a sequence departing from and returning to one of the airline's crew bases. In order to be flyable, a rotation must comply with all of the relevant federal, company, and union regulations; e.g., the crew must return to its base within three days. Since these regulations can be specified exactly, many thousands of flyable rotations can be generated rapidly by computer. A cost is calculated for each rotation, in accordance with the airline's contract with the pilot's union. Once a complete set J of flyable rotations has been generated, the problem is to select out an optimal subset. A subset J^1 is feasible if every flight segment belongs to exactly one of the rotations in J^1. An optimal subset is a feasible subset with minimal cost.

Formally, let $I = \{1,\ldots,m\}$ index the flight segments and $J = \{1,\ldots,n\}$ index the given set of flyable crew rotations. Define an incidence matrix $A = (a_{ij})$ by

$$a_{ij} = \begin{cases} 1 & \text{if flight segment } i \text{ is on rotation } j \\ 0 & \text{otherwise} \end{cases}$$

and let c_j denote the cost of rotation j. Then (PP) is a mathematical statement of the problem just described, provided we make the obvious interpretation:

$$y_j = \begin{cases} 1 & \text{if rotation } j \text{ is selected for } J^1 \\ 0 & \text{otherwise} \end{cases}$$

Unfortunately, even the plethora of federal, company, and union regulations still allows a very large number of flyable rotations. Thus each of the airlines that contributed problems to this study also incorporated heuristic rules in its rotation generator. These rules were derived by the manual schedulers and reflect their conception of what a "good" rotation ought to look like. Under these heuristics J becomes a set of "acceptable" rotations.

The set partitioning problem was deemed worthy of intensive study because of its many applications and because its special structure promises that useful results can be obtained for very large problems. The goal of this study was to devise an algorithm for partitioning problems that are large enough to be of practical utility. This meant on the order a few hundred constraints in (PP) and several thousand binary variables. Problems of this magnitude have been beyond the scope of most special purpose algorithms previously developed, and far beyond the scope of general purpose integer programming algorithms. Furthermore, the special purpose algorithms have often been unsuccessful when any general linear constraints are appended to (PP). Such side conditions are very common in applications. In the airline case, for example, there are crew base constraints. These are of the form:

$$\text{(CB)} \qquad \sum_{j \in D_s} h_j y_j \leq M_s \quad \text{for } s=1,\ldots,q$$

where h_j is the number of flying hours, per month, associated with rotation j, M_s is the maximum number of flying hours available, per month, at crew base s, and D_s is the set of rotations flown out of crew base (or domicile) s.

The goal of this research, then, was to devise a special algorithm for problem (PP) which would be able to solve large problems in a reasonable amount of time, even in the presence of a limited number of side conditions.

1.2 Other Approaches

Several other algorithms for set covering and partitioning problems have appeared recently. Those which have had significant computational success will be mentioned here. Note that all of them begin by relaxing the binary condition on y (i.e. allowing $0 \leq y \leq 1$) and solving the resulting linear program. This is true of the present algorithm as well. The differences lie in the method of getting from the continuous optimum to the discrete optimum. The three alternatives that have been tried are cutting planes, group theory, and enumeration. While all three have been successful, enumeration has a clear advantage when there are side conditions. Both the cutting plane and the group theory approaches work best when the determinant of the optimal LP basis is small. This is usually the case for pure covering and partitioning problems. Appending general linear constraints, however, often causes this determinant to become very large.

Fortunately, the same test problem has been solved by one algorithm from each class: cutting plane, group theoretic, and enumerative. This problem, to be called UA1, was generated by United Airlines. It is a partitioning problem (i.e. equality constraints) with 117 rows and 4,845 binary variables. There are no crew base constraints. The present author's enumerative algorithm (to be presented in Section 3) solved problem UA1 in 8 minutes on an IBM 360/91. (For this type of program the Model 91 is about twice as fast as a Model 65.)

Glenn Martin of Control Data Corporation has solved a great many covering and partitioning problems with a proprietary cutting plane algorithm. He reports [28] that a typical moderate-sized airline crew scheduling problem of about 100 inequality constraints and 4,000 binary variables takes on the order of 10 minutes to solve on a CDC 3600. A typical larger problem of 150 rows and 7,000 binary variables can often be solved with a few cuts in the order of 40 minutes of CDC 3600 time. Note that these figures are for covering problems. Problem UA1, with equality constraints, took 55 minutes on the CDC 3600. This suggests that partitioning problems may be considerably harder to solve than covering problems, at least for a cutting plane algorithm. We shall return to this point later. Attempts to solve problems with crew base constraints have apparently been unsuccessful.

Thiriez [40] has developed a group theoretic algorithm based on the ideas in [36,37]. His computational experience is somewhat difficult to interpret since for large problems he uses a "semi automatic inspection" procedure in which part of the problem is "solved by visual inspection rather than by a program." This presumably involves recognition of simple group structures. He has solved several covering and partitioning problems, again of the airline crew scheduling type. Problem UA1 was solved in 22 minutes of IBM 360/65 time, plus an unspecified amount of "visual inspection" time. No results are reported for problems with crew base constraints.

A purely enumerative algorithm for the partitioning problem was developed by Pierce [30]. Without any help from linear programming, problems with up to a dozen equality constraints and several hundred binary variables were solved in a few seconds. More recently [32], linear programming has been incorporated so that larger problems could be handled.

For example, the new algorithm solved a truck routing problem with 60 equality constraints and 3,316 binary variables in about 11 minutes on an IBM 360/67.

Another enumerative algorithm for both covering and partitioning problems which uses linear programming has been proposed by Lemke, Salkin, and Spielberg, [25]. Their results include a partitioning problem with 50 equality constraints and 905 binary variables that was solved in 11 minutes on an IBM 360/50. The test problems used were of diverse origins, many being simply randomly generated.

The algorithm to be developed here differs from the other enumerative algorithms mentioned above in that it does not proceed in terms of decisions about individual variables. That is, the basic choice involved is not whether a certain variable should be used or not used (i.e., which specific variable should cover a particular row). Instead, the variables are grouped together in classes and the basic choice involved is which class should be responsible for covering a particular row. The status of individual variables is then determined automatically. The very large number of variables present in any realistic partitioning problem makes this approach quite attractive. Since essential use will be made of the equality constraints, however, the resulting algorithm will be confined to the partitioning problem and will not be able to solve the slightly more general covering problem. This specialization was regarded as worthwhile for two reasons. First, the partitioning problem is of great interest in its own right. The variety of applications mentioned in Section 1.1 above makes this clear. In some instances the partitioning and covering models both apply and represent slight variations in the underlying problem. This is the case for airline crew scheduling. In other applications only

the partitioning model is relevant. For example, in political districting each census tract must belong to exactly one voting district. Second, it was believed that much larger partitioning than covering problems could be solved -- this because of the severity of the equality constraints. The validity of this conjecture will be discussed in Section 6.

1.3 Reformulation

In most of the remainder of this paper it will be more convenient to discuss problem (PP) in terms of sets and partitions than in terms of constraints and variables. Let us begin by giving the set theoretic statement of the problem. Let $I = \{1,\ldots,m\}$ be the row index set and $J = \{1,\ldots,n\}$ be the column index set. Then

$$A_j \equiv \{i \in I \mid a_{ij} = 1\} \qquad \text{for } j \in J \tag{1.1}$$

is the subset of I containing exactly those rows covered by column j. A subset $J^1 \subseteq J$ is called a covering of I if

$$\bigcup_{j \in J^1} A_j = I \ . \tag{1.2}$$

Any covering of I is a partition of I if the corresponding sets are mutually disjoint:

$$j,k \in J^1 \text{ and } j \neq k \text{ implies } A_j \cap A_k = \emptyset \ . \tag{1.3}$$

Let F denote all those partitions of I that can be obtained from the columns of the matrix A,

$$F = \{J^1 \subseteq J \mid J^1 \text{ satisfies (1.2) and (1.3)}\}. \tag{1.4}$$

Then problem (PP) can be stated as:

(PP) Choose $J^1 \in F$ so as to minimize $\sum_{j \in J^1} c_j$.

The algorithm developed below will perform an implicit enumeration of the set F.

2. The Enumerative Scheme

The task in this section is to derive an efficient scheme for exhaustively enumerating the set F of feasible solutions to problem (PP). This scheme will serve as the framework for the implicit enumeration to be introduced in Section 3. The approach will be to show that F is in one-to-one correspondence with another set, $\overline{G}$, for which there is a very natural procedure.

2.1 Derivation of $\overline{G}$

The key idea to be exploited here is that the sets A_j can be grouped into classes in such a way that no two sets from the same class can appear together in any member of F. If B is any subset of J, then we shall call B an interference class if

$$j,k \in B \text{ and } j \neq k \text{ implies } A_j \cap A_k \neq 0 .$$

If B is an interference class and $J^1 \in F$, then the set $J^1 \cap B$ must either be a singleton or else be empty. For suppose that $j,k \in J^1 \cap B$ with $j \neq k$. Then by the above definition we see that

$$A_j \cap A_k \neq \emptyset$$

and hence that $\{A_j \mid j \in J^1\}$ cannot be a partition of I. A sufficient condition for $B \subseteq J$ to be an interference class is:

$$\bigcap_{j \in B} A_j \neq \emptyset \quad . \tag{2.1}$$

The next step is to partition J into interference classes. This can be done in a variety of ways, any of which will suffice. For example, define

(2.2) $$e_j \equiv \min\{i \in I \mid a_{ij} = 1\}$$

(2.3) $$E_t \equiv \{j \in J \mid e_j = t\} .$$

Then

(2.4) $$J = \bigcup_{t=1}^{m} E_t$$

where each non-empty E_t is an interference class, since

(2.5) $$\bigcap_{j \in E_t} A_j \supseteq \{t\} \neq \emptyset$$

and hence E_t satisfies the sufficient condition (2.1). It can therefore be assumed that a partition

(2.6) $$J = \bigcup_{t \in T} B_t$$

is given, where $T = \{1,\ldots,p\}$ and each B_t is non-empty and an interference class. The B_t will be referred to as <u>blocks</u>. As demonstrated by the example above, we can assume that $p \leq m$. By the argument given above it follows that if $J^1 \in F$, then for each $t \in T$ the set $J^1 \cap B_t$ is either a singleton or empty.

Now consider the class of functions g which map the set I into the set T. Denote this class by G,

(2.7) $$G \equiv \{g: I \to T\} .$$

An element of G assigns each row to a specific block (interference class). Each function g also induces a partition of I, since

(2.8) $$I = \bigcup_{t \in T} g^{-1}(t)$$

for any $g \in G$, where

(2.9) $$g^{-1}(t) \equiv \{i \in I \mid g(i) = t\}$$

i.e. $g^{-1}(t)$ is the set of rows which are assigned to block t by g. The

fact that each function g induces a partition of I suggests a relationship between the elements of G and those of F.

A one-to-one correspondence between the set of feasible solutions F and a subset of G can be established in the following manner. Let $g \in G$. For each $t \in T$ look at the rows which have been assigned to B_t by g; this is $g^{-1}(t)$. If there is at least one such row (i.e. if $g^{-1}(t) \neq \emptyset$), check to see if B_t contains a column j for which $A_j = g^{-1}(t)$. If such a column can be found for every $t \in T$ with $g^{-1}(t) \neq \emptyset$, then the collection of these columns is a partition of I. To formalize this, define

$$B_t(g) \equiv \{j \in B_t \mid A_j = g^{-1}(t)\} \tag{2.10}$$

for each $g \in G$ and $t \in T$. Notice that this set is either a singleton or empty. For column j to belong, A_j must exactly match $g^{-1}(t)$. (It can be assumed without loss of generality that all columns are unique.)

Definition (2.10) permits characterization of the subset of G which is in one-to-one correspondence with F, namely

$$\bar{G} \equiv \{g \in G \mid B_t(g) \neq \emptyset \text{ for all } t \text{ with } g^{-1}(t) \neq \emptyset\} . \tag{2.11}$$

The one-to-one correspondence is easily demonstrated:

i) Given $g \in \bar{G}$, let

$$J^1 = \bigcup_{t \in T} B_t(g) .$$

Since $g \in \bar{G}$ we know that $B_t(g)$ is a singleton, $\{j_t\}$, for each $t \in T^1$, where

$$T^1 = \{t \in T \mid g^{-1}(t) \neq \emptyset\} .$$

Therefore

$$J^1 = \{j_t \mid t \in T^1\}$$

and

$$A_{j_t} = g^{-1}(t) \quad \text{for } t \in T^1 .$$

It follows that

$$I = \bigcup_{t \in T} g^{-1}(t)$$

$$= \bigcup_{t \in T^1} g^{-1}(t)$$

$$= \bigcup_{t \in T^1} A_{j_t}$$

$$= \bigcup_{j \in J^1} A_j \quad .$$

The disjointness of the sets A_j for $j \in J^1$ follows from that of the sets $g^{-1}(t)$ for $t \in T^1$. Thus $J^1 \in F$.

ii) Let $J^1 \in F$ so that

$$\bigcup_{j \in J^1} A_j = I \ .$$

Define g on I by defining it on each A_j. For each $j \in J^1$, let g map all of the rows in A_j onto the block which contains column j. If $g^{-1}(t) \neq \emptyset$, then

$$g^{-1}(t) = A_{j^*} \text{ where } \{j^*\} = J^1 \cap B_t$$

so that

$$B_t(g) = \{j^*\} \neq \emptyset \ .$$

Therefore $g \in \overline{G}$.

In terms of the g-functions, problem (PP) can be restated as:

(PP) minimize cost (g) subject to $g \in \overline{G}$,

where cost (g) $\equiv \sum_{j \in J^1} c_j$

and $J^1 = \bigcup_{t \in T} B_t(g)$.

2.2 Enumeration of $\overline{G}$

The first purpose of this section has now been accomplished. The set $\overline{G}$ has been defined and shown to be in one-to-one correspondence with the set F. Enumerating the functions in $\overline{G}$ is therefore equivalent to enumerating the partitions in F. The specification of a function $g \in G$ can be viewed as an m-stage decision process: at stage i, g(i) is chosen from among the elements of T. An enumeration of these functions can be confined to $\overline{G}$ by suitably restricting the range of choice at each stage. That is, if g(1), ...,g(r) have already been determined, then g(r+1) will be chosen from a certain set $T^* \subseteq T$ rather than from the entire set T. As a result, every complete mapping g(1),...,g(m) enumerated will belong to $\overline{G}$. The method of determining the set T^* is the next order of business.

The definitions introduced above will now be generalized in a very straightforward way. Let $1 \leq r \leq m$ so that

$$\{1,\ldots,r\} \subseteq I$$

and define

$$G^r \equiv \{g_r : \{1,\ldots,r\} \to T\} \tag{2.12}$$

$$A_j^r \equiv A_j \cap \{1,\ldots,r\} \qquad \text{for all } j \in J . \tag{2.13}$$

Each $g_r \in G^r$ is called an r-partial function since it maps only the first r rows of I into T. Definitions (2.10) and (2.11) are generalized to

$$B_t^r(g_r) \equiv \{j \in B_t \mid A_j^r = g_r^{-1}(t)\} \tag{2.14}$$

$$\overline{G}^r \equiv \{g_r \in G^r \mid B_t^r(g_r) \neq \emptyset \text{ for all t with } g_r^{-1}(t) \neq \emptyset\}. \tag{2.15}$$

The set $B_t^r(g_r)$ contains all those columns of B_t that match $g_r^{-1}(t)$, up to row r. Note that when r = m, $B_t^r(g_r)$ must either be a singleton or empty. When $r < m$, however, $B_t^r(g_r)$ may contain several columns. Clearly

$$G = G^m \text{ and } \overline{G} = \overline{G}^m \quad . \tag{2.16}$$

The problem of enumerating only those complete (m-partial) mappings which belong to $\bar{G}$ will now be reduced to a simpler problem. Let us make the convention that $G^0 = \{\emptyset\}$. Given an arbitrary $g_r \in \bar{G}^r$, where $0 \leq r \leq m$, it now suffices to know which (r+1)-partial extensions of g_r belong to $\bar{G}^{r+1}$. To see how this may be determined, let h be an extension of g_r:

$$\begin{aligned} h(i) &= g_r(i) \quad \text{for } i=1,\ldots,r \\ h(r+1) &= t^* \quad . \end{aligned} \tag{2.17}$$

What conditions on the choice of t^* will ensure that $h \in \bar{G}^{r+1}$? From (2.17) it follows that

$$h^{-1}(t^*) = g_r^{-1}(t^*) \cup \{r+1\} \tag{2.18}$$

$$h^{-1}(t) = g_r^{-1}(t) \quad \text{for } t \neq t^* \quad . \tag{2.19}$$

We must consider the sets $B_{t^*}^{r+1}(h)$ and $B_t^{r+1}(h)$ for $t \neq t^*$.

$$\begin{aligned} B_{t^*}^{r+1}(h) &= \{j \in B_{t^*} \mid A_j^{r+1} = h^{-1}(t^*)\} \\ &= \{j \in B_{t^*} \mid A_j^{r+1} = g_r^{-1}(t^*) \cup \{r+1\}\} \\ &= \{j \in B_{t^*} \mid A_j^r = g_r^{-1}(t^*) \quad \text{and} \quad r+1 \in A_j\}. \end{aligned}$$

Therefore $B_{t^*}^{r+1}(h)$ contains all of the columns of $B_{t^*}^r(g_r)$ which also cover row (r+1).

$$B_{t^*}^{r+1}(h) = B_{t^*}^r(g_r) \cap R_{r+1}^1 \tag{2.20}$$

where, for each $i \in I$,

$$R_i^1 \equiv \{j \in J \mid a_{ij} = 1\}. \tag{2.21}$$

Thus R_{r+1}^1 is just the set of all columns which cover row (r+1). Now consider any $t \in T$, $t \neq t^*$, for which $g_r^{-1}(t) \neq \emptyset$.

$$B_t^{r+1}(h) = \{j \in B_t \mid A_j^{r+1} = h^{-1}(t)\}$$

$$= \{j \in B_t \mid A_j^{r+1} = g_r^{-1}(t)\}$$

$$= \{j \in B_t \mid A_j^{r} = g_r^{-1}(t) \text{ and } r+1 \notin A_j\}.$$

$B_t^{r+1}(h)$ contains all of the columns of $B_t^r(g_r)$ which do not cover row (r+1). Defining, for $i \in I$,

$$R_i^0 \equiv \{j \in J \mid a_{ij} = 0\} = J - R_i^1 \tag{2.22}$$

gives us

$$B_t^{r+1}(h) = B_t^r(g_r) \cap R_{r+1}^0 \tag{2.23}$$

for $t \neq t^*$. The conditions that must be imposed on t^* follow immediately from (2.20) and (2.23):

$$B_{t^*}^r(g_r) \cap R_{r+1}^1 \neq \emptyset \tag{C1}$$

$$B_t^r(g_r) \cap R_{r+1}^0 \neq \emptyset \text{ for all } t \neq t^* \tag{C2}$$
$$\text{with } g_r^{-1}(t) \neq \emptyset .$$

These conditions ensure that $B_t^{r+1}(h) \neq \emptyset$ for all t with $h^{-1}(t) \neq \emptyset$ and hence that $h \in \bar{G}^{r+1}$.

To summarize this result: let $g_r \in \bar{G}^r$ where $0 \leq r \leq m$ and let h be an (r+1)-partial extension of g_r as in (2.17). Then $h \in \bar{G}^{r+1}$ if and only if

$$h(r+1) \in T(g_r) \tag{2.24}$$

where

$$T(g_r) \equiv \{t^* \in T \mid t^* \text{ satisfies (C1) and (C2)}\}. \tag{2.25}$$

A compact enumerative scheme for $\bar{G}$, and hence for F, can now be constructed. See, for example, the finite map enumerator of Graves and Whinston [19].

3. Implicit Enumeration

A means of shortcutting the complete enumeration of the previous section is essential if problems of any practical size are to be solved. The derivation of permissible shortcuts is therefore the task of the present section. The development will draw heavily upon the discussion of relaxation and fathoming by Geoffrion and Marsten [17].

3.1 Candidate Problems

Suppose that some part (or none) of the enumeration has already been performed and that g* is the best (i.e. cheapest) element of $\bar{G}$ that has been found so far. The function g* will be called the incumbent and we set V = cost(g*). If there is no incumbent as yet, then V = ∞. V is referred to as the ceiling, since it is an upper bound on the optimal value of (PP). Suppose that some r-partial map $g_r \in \bar{G}^r$, where $0 \leq r \leq m$, has just been constructed by the enumerative procedure. Let this g_r be fixed throughout the remainder of the discussion in this section.

The map $h_s \in \bar{G}^s$ is a feasible extension of g_r if $s > r$ and

$$h_s(i) = g_r(i) \quad \text{for } i=1,\ldots,r. \tag{3.1}$$

When s = m we have the feasible completions of g_r, denoted $C(g_r)$.

$$C(g_r) \equiv \{g \in \bar{G} \mid g(i) = g_r(i) \text{ for } i=1,..,r\} . \tag{3.2}$$

The candidate problem associated with g_r must now be introduced. If h_s is any feasible extension of g_r, then

$$h_s^{-1}(t) \supseteq g_r^{-1}(t) \quad \text{for all } t \in T \tag{3.3}$$

which implies that

(3.4) $\quad B_t^s(h_s) \subseteq B_t^r(g_r) \quad$ for all $t \in T$

As a special case of (3.4) we have

(3.5) $\quad B_t(g) \subseteq B_t^r(g_r) \quad$ for all $t \in T$

for every feasible completion g of g_r, $g \in C(g_r)$. Recall that for any $g \in \bar{G}$ the corresponding element of F is given by

(3.6) $\quad J^1 = \bigcup_{t \in T} B_t(g)$.

It follows from (3.5) and (3.6) that

(3.7) $\quad J^1 \subseteq \bigcup_{t \in T} B_t^r(g_r)$.

Therefore column j will not be selected by any completion of g_r unless it belongs to the set $J_a(g_r)$, where:

(3 8) $\quad J_a(g_r) \equiv \bigcup_{t \in T} B_t^r(g_r)$.

The set $J_a(g_r)$ contains all of the <u>available</u> columns, given the partial map g_r. Problem (PP) reduces to the following residual problem which will be called the candidate problem for g_r.

(PP^{g_r}) minimize cost(g) subject to $g \in C(g_r)$.

As an integer linear program this is:

$$(PP^{g_r}) \quad \text{minimize} \sum_{j \in J_a(g_r)} c_j y_j$$

$$\text{subject to} \sum_{j \in J_a(g_r)} a_{ij} y_j = 1 \quad \text{for } i \in I$$

$$y_j = 0 \text{ or } 1 \text{ for } j \in J_a(g_r) \ .$$

This is the same as (PP) except that "$j \in J_a(g_r)$" has replaced "$j \in J$".

By dropping the integrality conditions (PP^{g_r}) is relaxed to the ordinary linear program:

$$(LP^{g_r}) \quad \text{minimize} \sum_{j \in J_a(g_r)} c_j y_j$$

$$\text{subject to} \sum_{j \in J_a(g_r)} a_{ij} y_j = 1 \text{ for } i \in I$$

$$0 \leq y_j \leq 1 \text{ for } j \in J_a(g_r).$$

This linear program provides sufficient conditions under which g_r does not have to be extended. Using $F(\cdot)$ and $v(\cdot)$ to denote feasible region and optimal value, respectively, these conditions are:

1) If $F(LP^{g_r}) = \emptyset$, then $C(g_r) = \emptyset$.
2) If $v(LP^{g_r}) \geq V$, then $\text{cost}(g) \geq V$ for all $g \in C(g_r)$.
3) If an optimal solution of (LP^{g_r}) is naturally integer, then an optimal completion of g_r is known.

The enumerative scheme of Section 2 is easily modified to incorporate these fathoming tests based on linear programming. These modifications will be made after some additional improvements are introduced.

3.2 Penalties

Suppose that none of the conditions derived above are satisfied, so that g_r must be extended. In Section 2 the set $T(g_r)$ of those blocks eligible to receive row (r+1) was derived. A branch in the enumeration

tree is created for each $t \in T(g_r)$. There is still the question of which branch to explore first. This question will now be addressed with the aid of the penalty concept.

For a discussion of the use of penalties in integer programming, the reader should consult Driebeek [13], Davis, Kendrick and Weitzman [11], Tomlin [41], or Geoffrion and Marsten [17].

The aim is to select a path through the tree that leads to an optimal solution as quickly as possible. This enables the ceiling V to be lowered and thus speeds up the remainder of the enumeration. Among the branches descending from any node, the one that seems most likely to lead to an optimal solution should be explored first. Penalties provide a quantitative basis for such choices and enable us to follow a "path of least resistance" through the tree.

Let $t \in T(g_r)$ and let h be the (r+1)-partial extension of g_r for which $h(r+1) = t$. There is a penalty associated with the creation of h, to be denoted PEN(h), which will now be derived. If (LP^{g_r}) were infeasible, then g_r would not be extended. (LP^{g_r}) cannot have an unbounded solution, since its objective function is bounded below by zero. Hence it can be assumed that an optimal solution $y(g_r)$ of (LP^{g_r}) is at hand. Define

(3.9) $J_b(g_r) \equiv$ the set of columns which are basic in $y(g_r)$.

(3.10) $J_{nb}(g_r) \equiv$ the set of columns which are non-basic in $y(g_r)$.

The partial map g_r therefore induces two different partitions of the set J. First,

$$J = J_a(g_r) \cup J_{na}(g_r) \tag{3.11}$$

separates J into those columns which are still available, $J_a(g_r)$, and the remainder which are not available, $J_{na}(g_r)$. Second,

(3.12) $$J = J_b(g_r) \cup J_{nb}(g_r)$$

divides the columns according to whether they are basic or non-basic in $y(g_r)$. From the definition of (LP^{g_r}) it is clear that

(3.13) $$J_b(g_r) \subseteq J_a(g_r).$$

i.e. only available columns can be made basic.

Now consider the extension h. Any column available under h must have been available under g_r,

(3.14) $$J_a(h) \subseteq J_a(g_r).$$

This follows from (3.4) and (3.8) with s=r+1. There are two cases to consider. If

(3.15) $$J_a(h) \supseteq J_b(g_r)$$

then it follows that $y(g_r)$ is an optimal solution of (LP^h) and hence

(3.16) $$v(LP^h) = v(LP^{g_r}) .$$

In this case define PEN(h) = 0. If, on the other hand, condition (3.15) is not satisfied, then

(3.17) $$J_{na}(h) \cap J_b(g_r) = \{j_1,..,j_k\} \neq \emptyset$$

that is, some of the columns that were basic under g_r are not available under h. A lower bound on the cost of adding the constraint

(3.18) $$y_{j_1} + \ldots + y_{j_k} = 0$$

to problem (LP^{g_r}) is given by the amount that the objective function would increase in one dual point. It is this amount that is denoted PEN(h) when (3.15) does not hold. In either of the above cases

(3.19) $$v(LP^h) \geq v(LP^{g_r}) + PEN(h)$$

holds. Since

(3.20) $$v(PP^h) \geq v(LP^h) ,$$

a lower bound on the optimal value of the candidate problem for h has been obtained:

$$v(PP^h) \geq v(LP^{g_r}) + PEN(h) \quad . \tag{3.21}$$

These lower bounds play an important role in the algorithm. Define

$$BOUND(h) \equiv v(LP^{g_r}) + PEN(h) \tag{3.22}$$

for each extension $h \in \bar{G}^{r+1}$ of $g_r \in \bar{G}^r$.

The extensions of g_r can now be considered in order according to their bounds, the one with the lowest bound being selected first. Schematically, the branches descending from the node for g_r will be drawn with the value of BOUND increasing from left to right. The branches will then be explored in that order.

The bounds derived above not only serve to guide the search, they also provide new opportunities for fathoming. No extension need ever be pursued unless its bound is below the <u>current</u> ceiling V. For if

$$BOUND(h) \geq V \tag{3.23}$$

then it is clear from (3.21) and (3.22) that h need not be considered. This mechanism for fathoming h <u>without solving (LP^h)</u> turns out to be extremely effective computationally.

3.3 The Algorithm

The final form of the algorithm for the pure set partitioning problem can now be given. A logical flowchart is displayed in Figure 1. In the statement of the algorithm, g refers to the current partial map: g(1), g(2),...,g(i).

The Algorithm

Step 0. Set i=0, g = $\emptyset$, BOUND($\emptyset$) = 0, $T_0 = \emptyset$ and V = ∞ .

Step 1. If (LP^g) is infeasible, go to Step 10.

Step 2. If $v(LP^g) \geq V$, go to Step 10.

Step 3. If the optimal solution obtained for (LP^g) is naturally integer, go to Step 9.

Step 4. If i=m, go to Step 9.

Step 5. Set i=i+1 .

Step 6. If T(g) = $\emptyset$, go to Step 13.

Step 7. Compute BOUND(h) for each $h \in T(g)$ and sort T(g).

Step 8. Set $T_i = T(g)$, go to Step 11.

Step 9. Record the solution. Set $V = \min\{V, v(LP^g)\}$.

Step 10. If $T_i = \emptyset$, go to Step 13.

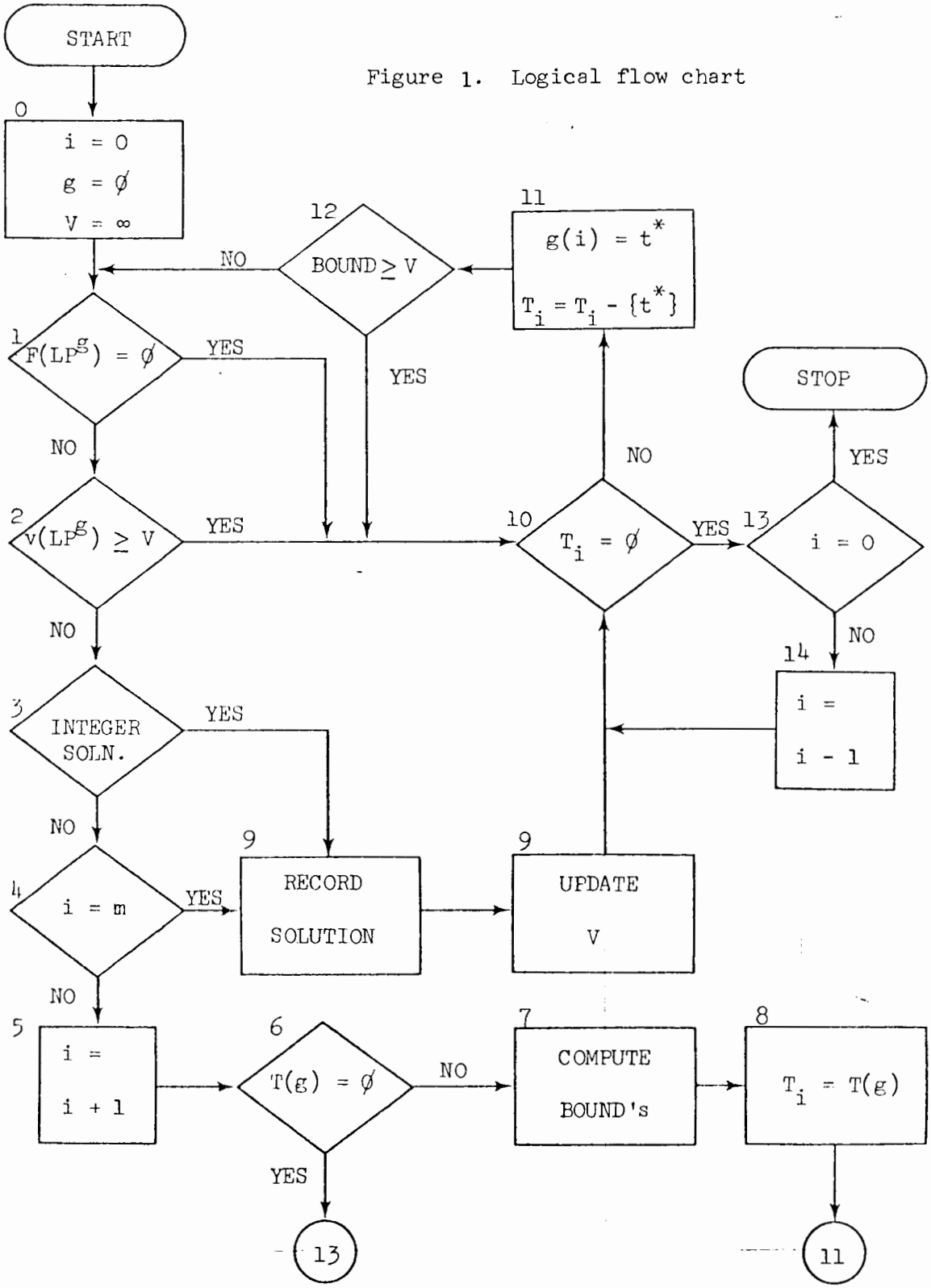
Step 11. Let t* = the first element in T_i. Let g(i) = t* and $T_i = T_i - \{t^*\}$.

Step 12. If BOUND(g) $\geq$ V, go to Step 10. Otherwise go to Step 1.

Step 13. If i=0, stop.

Step 14. Set i=i-1, go to Step 10.

Figure 1. Logical flow chart



4. Computational Experience

All of the computational results to be reported in this section are for pure set partitioning problems. Results obtained with the modified algorithm for partitioning problems with side constraints will be presented in the following section.

4.1 Implementation

Space does not permit a detailed account of the implementation of the algorithm. The interference classes were defined as in (2.2) and (2.3). This is the "staircase form" used by Pierce [30]. The key problem is keeping track of the membership of $J_a(g_r)$. Although awkward to express mathematically, this can be programmed very efficiently. It is also important to note that once the initial linear program ($LP^{\emptyset}$) has been solved, all of the subsequent LP solutions are obtained by quick reoptimizations. The interested reader can consult [27]. It is possible to demonstrate by purely logical arguments that certain columns cannot appear in any feasible solution. When these columns are discarded, some of the rows may become identical and hence redundant. An algorithm for performing this logical reduction is given in [27]. The problem sizes quoted below are after logical reduction.

4.2 Results

The algorithm has performed very well on all of the problems solved to date. At first, small test problems were constructed by generating the zeros and ones of the constraint matrix (a_{ij}) as independent Bernoulli trials. All of the larger problems generated in this manner, however,

turned out to be infeasible. This made it necessary, as well as desirable, to experiment with real problems. The test problems to be reported here are all of the airline crew scheduling type and were provided by Air Canada (AC), American Airlines (AA), and United Airlines (UA). One consequence of using real data was that problem size was not under our control. Each problem solved was significantly larger than the preceeding one. While the enumerative procedure has remained the same, the linear programming code has undergone a major revision at each step. At the present time it is again the difficulty of solving the initial linear program that prevents us from moving up to still larger problems. An unsuccessful attempt to solve a 400-row problem will be discussed below.

The results for pure set partitioning problems are presented in Table 1. The following symbols are used for the column headings.

ID - problem identifier: airline initials and sequence number.

m - the number of rows (partitioning constraints).

n - the number of columns (binary variables).

t(lp) - the time, in seconds, required to solve the initial linear program.

piv(lp) - the number of (dual) pivots required to solve the initial linear program.

cost(lp) - the optimal cost for the initial linear program.

t(e) - the additional time, in seconds, required to perform the enumeration. That is, to find and verify the optimal integer solution.

piv(e) - the number of (dual) pivots required for reoptimization during the enumeration phase.

cost(e) - the cost of the optimal integer solution.

rmax - the maximum depth of the search, i.e. the maximum number of rows assigned at any one time.

nsol - the number of integer solutions found.

d - the density of the coefficient matrix.

Time spent on basis reinversions is included in t(lp) and t(e). The pivots performed during these reinversions are not counted in piv(lp) or piv(e). All of the times quoted in Table 1 are for the UCLA IBM 360/91.

Logical reduction was quite effective on problem AC1, reducing it in size from (142 x 544) to (90 x 303). Effectiveness appeared to diminish, however, with increasing size. The reduction for problem UA1 was only from (117 x 4,845) to (111 x 4,826).

Problem AA4 is a subset of problem AA3. A selection of about half of the rows of AA 3 was made. With half of the rows missing, many of the columns were no longer unique. Duplicates were screened out and the problem was then run with a set of unique columns. A 200 row problem was of special interest since it lies midway between the easy 100 row UA1 and the unsolved 400 row AA3. Problem AA4, unfortunately for our purposes, had a natural integer solution.

The linear programming code used in this study was a dual version of the primal algorithm developed by Graves [18]. A dual algorithm was used so as to avoid the massive primal degeneracy encountered in linear programs derived from covering and partitioning problems. Although the dual algorithm must scan <u>every</u> column at each iteration, it still ran much faster than the primal algorithm. This superiority can be attributed to the fact that the dual starts out feasible and makes a positive gain in its objective function at virtually every iteration. By contrast, the primal algorithm does not start out feasible and typically requires several

Table 1. Results for pure set partitioning problems. (360/91)

ID	m	n	t(lp)	piv(lp)	cost(lp)	t(e)	piv(e)	cost(e)	rmax	nsol	d
AC1	90	303	8.53	128	42,719.50	32.11	19	42,855.00	56	1	7%
AA1	63	1,641	84.50	326	60,902.50	84.26	201	60,990.00	12	2	11%
UA1	111	4,826	314.93	455	217,351.	164.84	122	217,687.00	19	3	5%
AA4	200	2,362	418.86	612	81,730.00	0.00	0	81,730.00	0	1	1½%
AA3	419	21,585	> 3 hrs.	?	> 119,800	?	?	?	?	?	2%

(often a great many) iterations to make a move toward feasibility or to achieve a reduction in its objective function.

Fortunately, the entire A matrix could be kept in main memory since only the locations of the relatively rare unit entries had to be stored. This eliminated the input/output operations that ordinarily make dual LP codes so slow for large problems.

Our attempt to solve the initial linear program for AA3 was terminated, very reluctantly, after a total of about three hours of CPU time on the Model 91. An earlier attempt, with IBM's latest mathematical programming system (MPSX), also failed. The MPSX-PRIMAL algorithm encountered the severe degeneracy characteristic of this problem type and got trapped. That is, it reached a point from which it could not depart within several hundred iterations. Even when a primal feasible starting basis was provided, the MPSX-PRIMAL algorithm had not escaped from this starting solution after 1000 iterations! Perturbations were tried, but to no avail.

Recall from Section 1 that all of the algorithms that have solved large partitioning problems begin by solving the associated linear program. It is clear that the relative difficulty of this class of linear programs represents a serious common bottleneck.

4.3 Example

In order to illustrate the behavior of the algorithm the entire enumeration tree for problem UA1 is displayed in Figure 2. Each node contains a row number. The branches descending from a model represent the blocks (interference classes) to which that row can be assigned. The branches are labelled accordingly, with block numbers. Each node is labelled with the corresponding value of BOUND. Numbers in parentheses

Figure 2 (a). The enumeration tree for problem UA1.

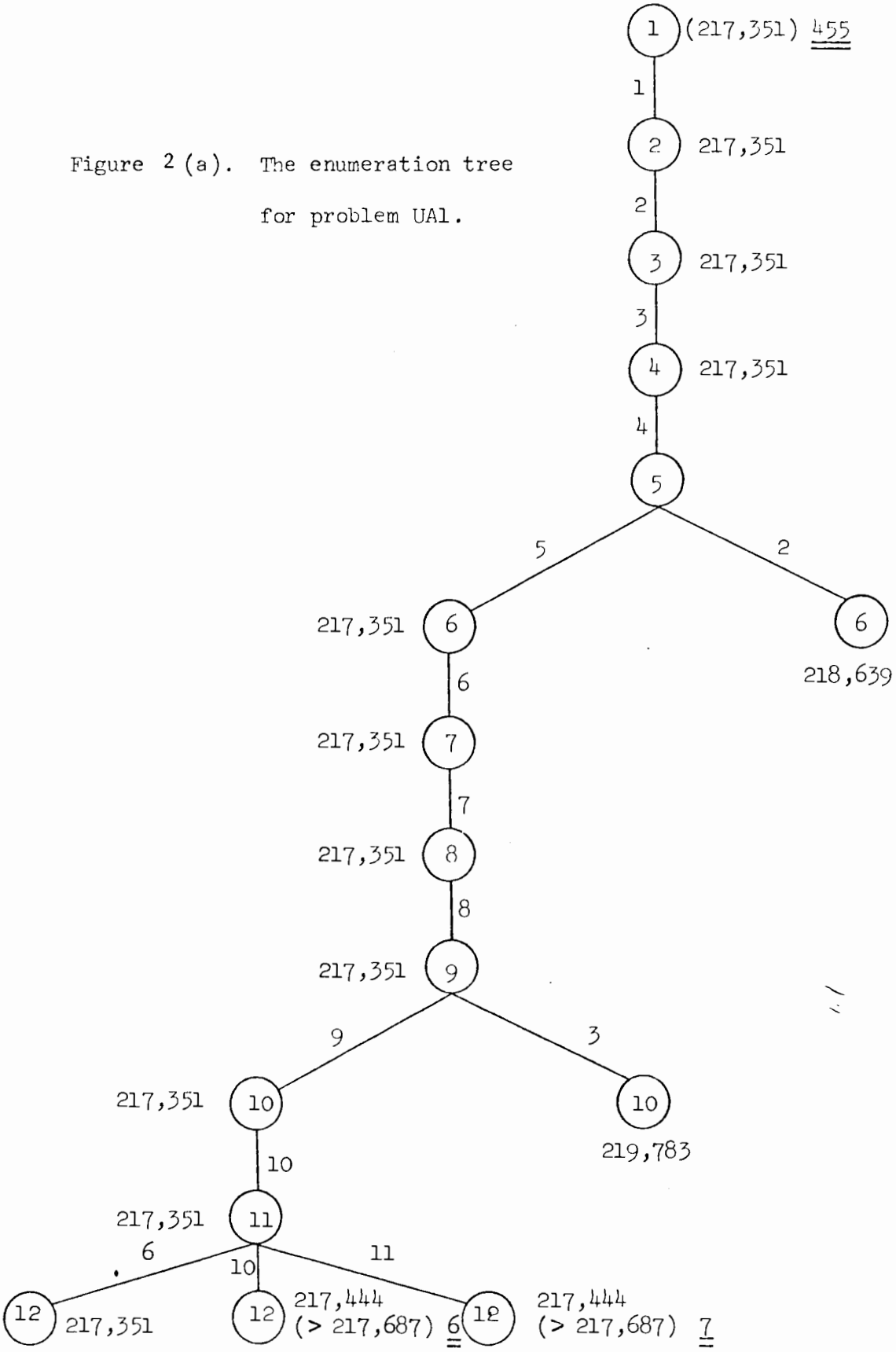


Figure 2 (b). The enumeration tree for problem UA1, cont'd.

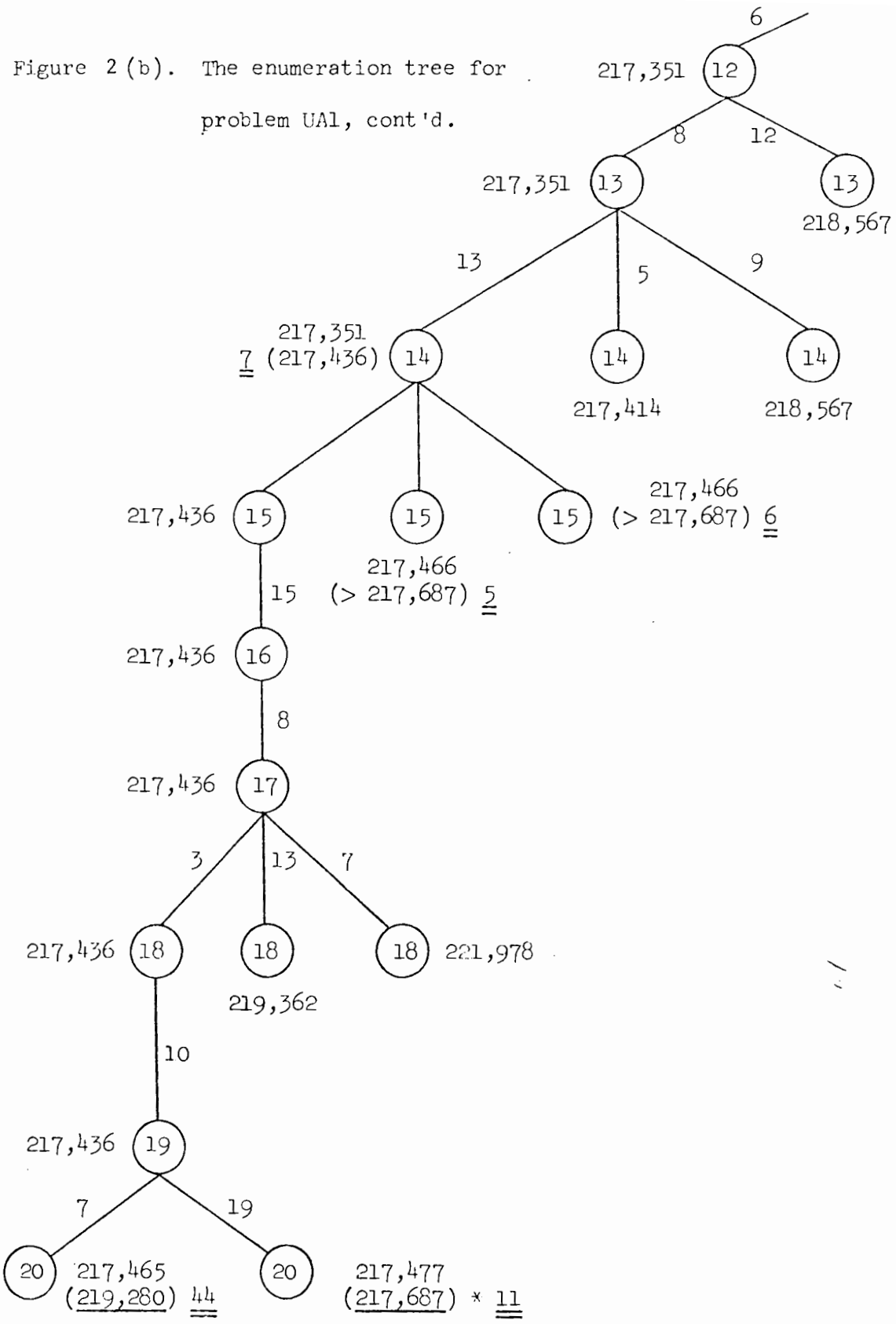
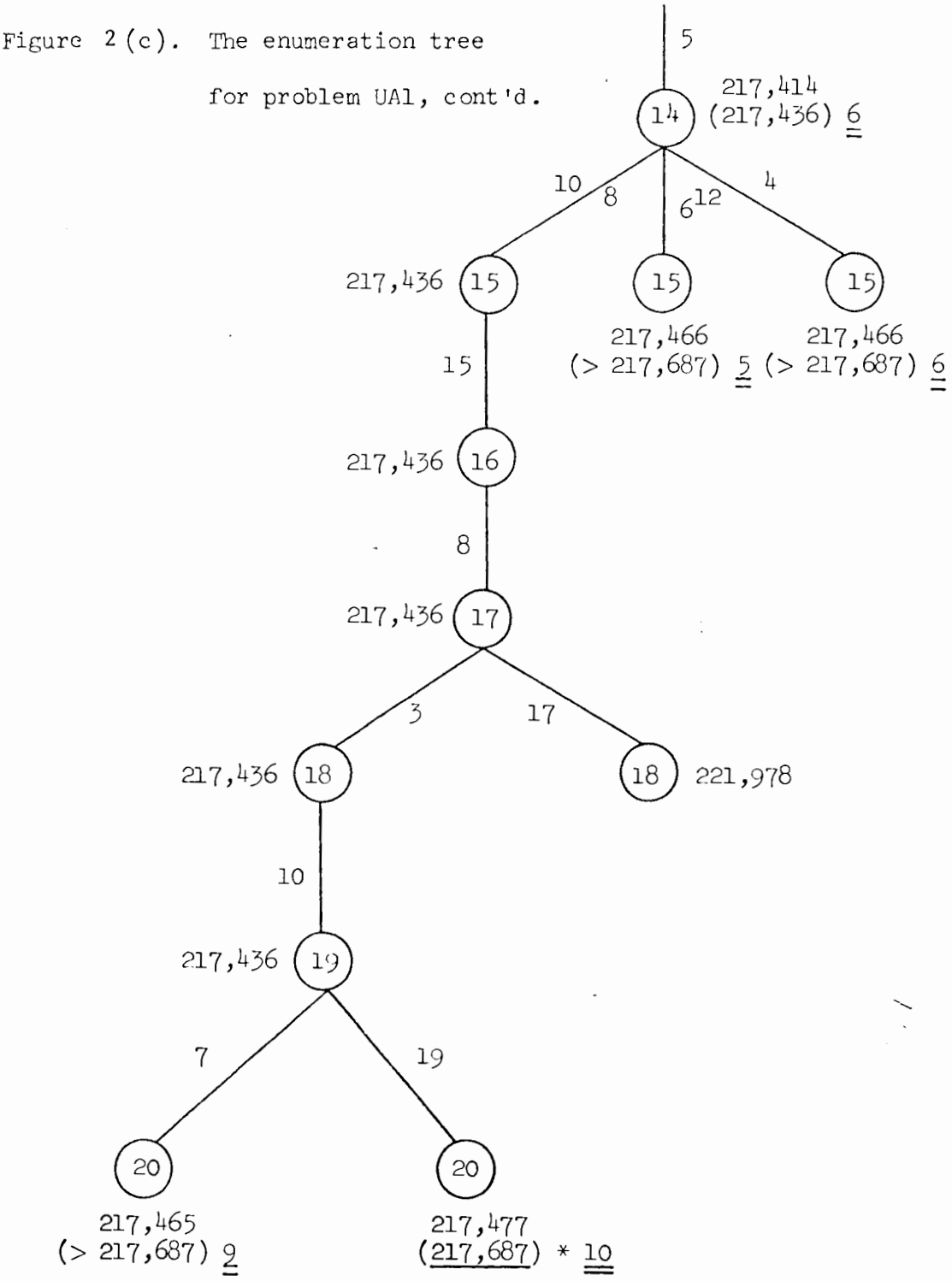


Figure 2 (c). The enumeration tree for problem UA1, cont'd.



represent values which are obtained by a complete, or partial, reoptimization of the LP tableau. Since reoptimization is performed by dual pivots it can be terminated as soon as the objective function exceeds the current ceiling V. An underlined number in parentheses indicates a natural integer solution. The numbers with double underlines are numbers of dual pivots.

4.4 Observations

The reasons for the excellent performance of the algorithm are fairly clear. Partitioning problems frequently have natural integer solutions - for example AA4. Since all of the candidate problems are partitioning problems in their own right, it is not surprising that the LP reoptimizations often lead to integer solutions. Of the four complete reoptimizations performed for problem UA1, three ended with integer solutions. For problem AA1 integer solutions were obtained two out of seven times, and for problem AC1 it was one out of one.

A second reason can be seen by comparing cost(lp) and cost(e) in Table 1: the continuous and integer minima are very nearly equal. In all cases they differ by less than ½% in value. This makes the penalties remarkably effective. Very few of the branches that are left for later consideration ever have to be explored. By the time we return to them, their lower bounds exceed the value of an integer solution already found. This is particularly true since the "path of least resistance" has in all cases led straight to an optimal, or very near optimal, solution.

The algorithm can be expected to work best on sparse matrices and to suffer from a proliferation of branches if the constraint matrix is very dense (say 50%). Table 2 shows the relationship between problem density

and the average number of branches created when one of the partial mappings was extended.

Table 2. Average Number of Descending Branches (b).

Problem	Density	b
UA1	5%	1.6
AC1	7%	1.5
AA1	11%	2.4

Very large problems, at least of the crew scheduling or truck delivery type, are always super-sparse (1-3%). The enumeration phase of the present algorithm should therefore work very well on such problems. In some applications, however, dense matrices may be the rule. In this event, the algorithm of Pierce and Lasky [32] would be preferable. Their algorithm works best on very dense matrices and has difficulties with sparse ones. The closely related algorithm of Garfinkel and Nemhauser [15] behaves in the same way.

If a collection of near optimal integer solutions is desired, these may be obtained by means of selective exploration of the enumeration tree after the optimal solution has been found. Reoptimizations usually require only a few dual pivots and can be counted on to produce integer solutions in a large proportion of cases. Such a set of near optimal solutions may be very valuable if other criteria in addition to cost are used to evaluate the feasible solutions.

5. Side Conditions

Side conditions of one form or another occur in many practical applications of the set partitioning model. For this reason the algorithm has been modified to handle side conditions on (PP).

5.1 Method

The side conditions are employed as filters. When an integer solution is found at Step 3 it is checked against the side conditions. If it satisfies them we proceed to Step 9 as usual. Otherwise, the current partial mapping must be extended. Note that side conditions can only hurt, that is, force more of the tree to be searched.

It has been assumed so far that each column of the constraint matrix A is unique. In the presence of side conditions this would involve a loss of generality since two columns of A might be identical and yet differ with respect to the side conditions. To allow for this, the following method is employed. One representative is chosen from each class of identical columns of A, namely the one with the lowest cost. The problem is solved as above with one exception. If an integer solution is found that does not satisfy the side conditions, then an attempt is made to exchange some of the representative columns for their duplicates in such a way as to satisfy the side conditions. This approach has the advantage that only unique columns appear in the linear program. This is important if there is extensive duplication, as in the problems discussed below. The full details of the method are contained in [27].

5.2 Results

Problems AC1 and AA1 were solved with crew base constraints of the form (CB) (see Section 1). The results are presented in Table 3. The column headings are as follows:

- m - number of rows.
- n' - the total number of columns of A.
- n - the number of unique columns of A.
- q - the number of crew base constraints.
- t'(e) - time (in seconds) for the enumeration phase, without crew base constraints.
- t(e) - time (in seconds) for the enumeration phase, with crew base constraints.

Times reported are for the IBM 360/91. The initial linear program is the same in both cases, i.e. with or without the crew base constraints.

Table 3 . Results for problems with crew base constraints (360/91)

ID	m	n'	n	q	t'(e)	t(e)
AC1	90	394	303	5	34.32	32.11
AA1	63	2,191	1,641	4	162.43	84.26

6. Conclusion

The computational results with pure set partitioning problems are certainly very encouraging. The solution times compare very favorably with those of other investigators. The main difficulty appears to lie in solving the initial linear program. As noted earlier, this difficulty is shared by all of the other serious computational approaches that have been tried so far.

Dramatic improvements might result if the special structure of the problem could be exploited during the linear programming computations. Balas and Padberg [2] have recently provided some clues in this direction. In particular, special properties of the LP bases might permit a fast reinversion procedure. The time currently being spent on basis reinversions is a very substantial part of the total computing time. These reinversions are being done by performing the required pivots in arbitrary order. Pre-selection of the pivoting sequence, as described by Kalan [22], could greatly reduce this burdensome overhead. Another promising idea would be to select a set of rows of the A matrix that do not overlap and treat them as generalized upper bounds [21]. This would allow the initial linear program to be solved faster and with a smaller inverse.

The success of the algorithm in the presence of side conditions is less clearcut. More experimentation will have to be done in this area. For linear side conditions, the most promising way to improve performance would be to include the side conditions in some of the linear programs and/or use them to compute additional penalties.

At the beginning of this study it was the author's opinion that partitioning problems should be much easier to solve than covering prob-

lems. While this is certainly true for purely enumerative algorithms (see, for example, [30] or [15]), it may not be true for LP-based algorithms. The severity of the partitioning constraints, which is the key to an efficient enumeration, is also the bane of the linear programming computations. Covering problems yield easier linear programs and also have an important advantage during the enumeration phase: fractional LP solutions can be rounded up to feasible integer solutions of a covering problem. This is done by Lemke, Salkin, and Spielberg [25]. Unfortunately, this convenient source of potential incumbents is not available for partitioning problems.

This study has demonstrated that in designing an implicit enumeration algorithm there are advantages to be reaped by making a judicious choice of what to enumerate. In the present case a special class of finite mappings was chosen, rather than the customary set of binary n-vectors. A similar approach has recently led to a very successful algorithm for the quadratic assignment problem [20]

REFERENCES

1. Arabeyre, J.P., J. Fearnley, F.C. Steiger and W. Teather, "The Airline Crew Scheduling Problem: A Survey," Transportation Science, 3, 2, 140-163, May, 1969.

2. Balas, E. and M.W. Padberg, "On the Set Covering Problem," Management Science Research Report No. 197, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, February, 1970.

3. Balinski, M.L., "Integer Programming: Methods, Uses, Computation," Management Science, 12, 3, 253-313, November, 1965.

4. Balinski, M.L. and R. Quandt, "On an Integer Program for a Delivery Problem", Operations Research, 12, 300-304, 1964.

5. Busacker, R.G. and T.L. Saaty, Finite Graphs and Networks, McGraw-Hill, New York, 1965.

6. Clarke, G. and S.W. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," Operations Research, 12, 4, 568-581, 1964.

7. Cobham, A., "A Statistical Study of the Minimization of Boolean Functions Using Integer Programming", Research Report RC-756, IBM Research Center, June, 1962.

8. Cobham A., R. Fridshal and J.H. North, "An Application of Linear Programming to the Minimization of Boolean Functions", Research Report RC-472, IBM Research Center, June, 1961.

9. Daly, W.N. and S.J. Spierer, Mobile Oil Corporation, Private Communication, July 11, 1969.

References (continued)

10. Dantzig, G.B. and J.H. Ramser, "The Truck Dispatching Problem," Management Science, 6, 1, 80-91, 1959.

11. Davis, R.E., D.A. Kendrick and M. Weitzman, "A Branch and Bound Algorithm for Zero-One Mixed Integer Programming Problems," Operations Research, 19, 4, 1036-1044, 1971.

12. Day, R.H., "On Optimal Extracting From A Multiple File Data Storage System: An Application of Integer Programming," Operations Research, 13, 3, 482-494, 1965.

13. Driebeek, N.J., "An Algorithm for the Solution of Mixed Integer Programming Problems," Management Science, 12, 7, 576-587, March, 1966.

14. Garfinkel, R.S., "Set Covering: A Survey", presented at the 17th International Converence of the Institute of Management Sciences, July, 1970.

15. Garfinkel, R.S. and F.L. Nemhauser, "The Set Partitioning Problem: Set Covering with Equality Constraints," Operations Research, 17, 5, 848-856, September-October, 1969.

16. Garfinkel, R.S. and G.L. Nemhauser, "Optimal Political Districting by Implicit Enumeration Techniques," Management Science, 16, 8, B495-508, 1970.

17. Geoffrion, A.M. and R.E. Marsten, "Integer Programming: A Framework and State-of-the-Art Survey", Management Science, 18, 9, 465-491, 1971.

18. Graves, G.W., "A Complete Constructive Algorithm for the General Mixed Linear Programming Problem", Naval Research Logistics Quarterly, 12, 1, March, 1965.

19. Graves, G.W. and A.B. Whinston, "A New Approach to Discrete Mathematical Programming", Management Science, 15, 3, 177-190, 1968.

References (continued)

20. Graves, G.W. and A.B. Whinston, "An Algorithm for the Quadratic Assignment Problem," Management Science, 16, 7, 453-471, March, 1970.

21. Grigoriadis, M.D., "A Dual Generalized Upper Bounding Technique", Management Science, 17, 5, 269-284, 1971.

22. Kalan, James E., "Naive Linear Programming", Technical Report CP-71009, Computer Science/Operations Research Center, Southern Methodist University, Dallas, Texas, May, 1971.

23. Kolner, T.N., "Some Highlights of a Scheduling Matrix Generator System", United Airlines, presented at the Sixth AGIFORS Symposium, Sept., 1966.

24. Lasky, J.S., "Optimal Scheduling of Freight Trucking", M.S. Thesis, Massachusetts Institute of Technology, June 1969.

25. Lemke, C.E., H.M. Salkin, and K. Spielberg, "Set Covering by Single Branch Enumeration with Linear Programming Subproblems", Operations Research, 19, 4, 998-1022, 1971.

26. Levin, A., "Fleet Routing and Scheduling Problems for Air Transportation Systems", Ph.D. Dissertation, Massachusetts Institute of Technology, January 1969.

27. Marsten, R.E., "An Implicit Enumeration Algorithm for the Set Partitioning Problem with Side Constraints", Ph.D. Dissertation, University of California, Los Angeles, October, 1971.

28. Martin, G.T., Control Data Corporation, Private communication, May 4, 1971.

29. Paul, M.C. and S.H. Unger, "Minimizing the Number of States in Incompletely Specified Sequential Functions", IRE Transactions on Electronic Computers, EC-8, 356-367, 1959.

30. Pierce, J.F., "Application of Comvinatorial Programming to a Class of All-Zero-One Integer Programming Problems", Management Science, 15,

191-209, 1968.

31. Pierce, J.F., "Pattern Sequencing and Matching in Stock Cutting Operations", Tappi, 53, 4, 668-678, April, 1970.

32. Pierce, J.F. and J.S. Lasky, "Improved Comvinatorial Programming Algorithms for a Class of All-Zero-One Integer Programming Problems," IBM Cambridge Scientific Center Report (to appear in Management Science), June, 1970.

33. Revelle, C., D. Marks, and J.C. Liebman, "An Analysis of Private and Public Sector Location Models", Management Science, 16, 12, 692-707, 1970.

34. Root, J.G., "An Application of Symbolic Logic to a Selection Problem", Operations Research, 12, 4, 519-526, 1964.

35. Salveson, M.E., "The Assembly Line Balancing Problem", Journal of Industrial Engineering, 6, 3, 18-25, 1955.

36. Shapiro, J.F., "Dynamic Programming Algorithms for the Integer Programming Problem - I. The Integer Programming Problem Viewed as a Knapsack Type Problem", Operations Research, 16, 1, 103-121, January-February, 1968a.

37. Shapiro, Jeremy F., "Group Theoretic Algorithms for the Integer Programming Problem II: Extension to a General Algorithm", Operations Research, 16, 5, 928-947, September-October, 1968b.

38. Spitzer, M., "Solution to the Crew Scheduling Problem", Presented at the First AGIFORS Symposium, October, 1961.

39. Steinmann, H. and R. Schwinn, "Computational Experience with a Zero-One Programming Problem", Operations Research, 17, 5, 917-920, 1969.

References (continued)

40. Thiriez, H., "Airline Crew Scheduling: A Group Theoretic Approach", Ph.D. Dissertation, Massachusetts Institute of Technology, October, 1969.

41. Tomlin, J.A., "Branch and Bound Methods for Integer and Non-Convex Programming", in J. Abadie (ed.), Integer and Nonlinear Programming, North-Holland, Amsterdam, 1970.

42. Valenta, J.R., "Capital Equipment Decisions: A Model for Optimal Systems Interfacing", M.S. Thesis, Massachusetts Institute of Technology, June 1969.

43. Wagner, W.H., "An Application of Integer Programming to Legislative Redistricting", CROND, Inc., Presented at the 34th National Meeting of ORSA, Philadelphia, Pennsylvania, November, 1968.