Automatic Identification of Generalized Upper Bounds in Large-Scale Optimization Models

Gerald G. Brown; David S. Thomen

Management Science, Vol. 26, No. 11 (Nov., 1980), 1166-1184.

Stable URL:
http://links.jstor.org/sici?sici=0025-1909%28198011%2926%3A11%3C1166%3AAIOGUB%3E2.0.CO%3B2-2

Management Science is currently published by INFORMS.

MANAGEMENT SCIENCE
Vol. 26, No. 11, November 1980
Printed in U.S.A.

AUTOMATIC IDENTIFICATION OF GENERALIZED UPPER BOUNDS IN LARGE-SCALE OPTIMIZATION MODELS*

GERALD G. BROWN† AND DAVID S. THOMEN†

To solve contemporary large-scale linear, integer and mixed integer programming problems, it is often necessary to exploit intrinsic special structure in the model at hand. One commonly used technique is to identify and then to exploit in a basis factorization algorithm a generalized upper bound (GUB) structure. This report compares several existing methods for identifying GUB structure. Computer programs have been written to permit comparison of computational efficiency. The GUB programs have been incorporated in an existing optimization system of advanced design and have been tested on a variety of large-scale real-life optimization problems. The identification of GUB sets of maximum size is shown to be among the class of NP-complete problems; these problems are widely conjectured to be intractable in that no polynomial-time algorithm has been demonstrated for solving them. All the methods discussed in this report are polynomial-time heuristic algorithms that attempt to find, but do not guarantee, GUB sets of maximum size. Bounds for the maximum size of GUB sets are developed in order to evaluate the effectiveness of the heuristic algorithms.
(PROGRAMMING; LARGE-SCALE SYSTEMS; GENERALIZED UPPER BOUNDS)

1. Introduction

Contemporary mathematical programming models are often so large that direct solution of the associated linear programming (LP) problems with the classical simplex method is prohibitively expensive, if not impossible in a practical sense. It has been found that most of these problems are sparse, with relatively few nonzero coefficients, and usually possess very systematic structure. These problems exhibit inherent structural characteristics that can be exploited by specializations of the simplex procedure.

Methods to exploit special model structure can be categorized generally as *indirect* (e.g., decomposition), where a solution to the original problem is achieved by dealing with related models which are individually easier to solve, or as *direct* when the original problem is solved by a modified simplex algorithm. Among the direct methods, the most frequently used technique is called *basis factorization* [7], where the reflection of special problem structure appears and is used to good benefit in the intermediate LP bases. Basis factorization can be *dynamic*, where the algorithm deals with each basis sequentially and/or independently in an attempt to extract as much specialized basis structure as possible, or *static*, where the algorithm depends upon certain types of special structure being present in *all* bases.

Static basis factorizations include *simple upper bounds, generalized upper bounds* (GUB), and *embedded network rows*, among many others. Simple upper bounds are a set of rows for which each row has only one nonzero coefficient. Generalized upper bounds are a set of rows for which each column (restricted to those rows) has at most

* Accepted by Arthur M. Geoffrion; received November 16, 1979. This paper has been with the authors 1 month for 1 revision.
† Naval Postgraduate School, Monterey.

0025-1909/80/2611/1166$01.25

one nonzero coefficient. Network rows are a set of rows for which each column (restricted to those rows) has at most two nonzero coefficients of opposite sign.

Each of these factorizations permits the simplex algorithm to deal with the static subsets of the rows (and columns) of all bases encountered with prior knowledge that they will satisfy *very* restricted rules. Most of these methods work best when *logic* can be substituted for arithmetic (as is the case with the coefficients ± 1). For this reason, static factorizations often restrict the special structure to possess only ± 1, or to be *scaled* so as to produce an equivalent result.

The concept of generalized upper bounds was introduced in 1964, the result of work by Dantzig and Van Slyke [5]. The name is derived from analogy to the simple upper bound structure. Graves and McBride [7] refer to *Static Signed Identity Factorization* as a term more suggestive of the implied basis structure. Since their introduction, some form of GUB has been implemented in many commercial LP systems. There is often confusion between the mathematical characterization of GUB and these various, widely used implementations of GUB, in that the latter often restrict the GUB set membership rules to permit uncomplicated simplex logic. All of the methods reported here address the full generality of GUB sets but can be modified to produce restricted GUB sets as necessary.

The details of how GUB can be exploited to reduce the computations of the simplex algorithm are not discussed here. See [1], [5], [7], [11], [13]. The underlying concept is that the GUB structure enables the simplex algorithm to manipulate the GUB rows implicitly, with logic rather than floating point arithmetic, thus reducing the effective size and solution time for the problem. The more GUB rows one is able to identify, the fewer rows one has to carry explicitly through the simplex operations. In large problems there exists a huge number of subsets of rows that satisfy the GUB criteria. It is generally regarded that those subsets with more rows are "better" GUB sets since they imply a more contracted explicit basis. The implied problem, then, is to find the *maximum* GUB set.

Optimal algorithms to find a maximum GUB set do exist. These entail enumeration schemes and cannot be guaranteed to be efficient in a practical sense. Conceivably, $2^m - m$ sets of rows might have to be searched before a maximum GUB structure is found: as the problem size grows, the number of possible sets that need to be checked increases *exponentially*. As will be shown later, the hope of finding an efficient algorithm to find the *maximum* GUB set for any general problem is dim.

Therefore, researchers and practitioners have concentrated on constructing efficient *heuristic* algorithms that attempt to identify, but do not guarantee, a maximum GUB set. A few of these methods showing great promise have been reported, but they have not been tested with large-scale problems.

This report (abstracted from [4]) outlines several automatic heuristic GUB-finding procedures that have been developed and published in the recent literature. These procedures are tested on a suite of large-scale, real-life optimization problems, and are modified to improve their behavior. Comparative performance of the methods is given both in terms of the computational effort to identify a GUB set, as well as the size of the GUB set achieved.

Identification of GUB sets of maximum row dimension is shown in Section 7 to be among the class of NP-complete problems. However, easily computed *upper bounds* on the size of the maximum GUB set are developed and used to evaluate objectively the

quality of heuristic GUB algorithms, showing that very nearly maximum GUB sets are routinely achieved.

2. Problem Definition and Representations

The Linear Programming problem is defined as

$$\begin{array}{ll} \text{Min} & c^t x \\ \text{s.t. } \underline{r} \leqslant Ax \leqslant \bar{r} & \text{(range constraints,)} \\ \underline{b} \leqslant x \leqslant \bar{b} & \text{(simple bounds),} \end{array} \tag{L}$$

where $\underline{r}$ and $\bar{r}$ are m-vectors, x, c, $\underline{b}$ and $\bar{b}$ are n-vectors and A is an $m \times n$ matrix. The constraints are sometimes defined as equations, but for the general case of GUB treated here constraints can be equations, inequalities or a mixture. The immediate discusssion will be directed at (L); integer and mixed-integer problems are treated later.

Two rows of A are said to *conflict* if there exists at least one column with nonzero coefficients in both rows. The GUB problem can be restated as that of finding a subset of the rows that do not conflict.

There are several ways one can model the maximum GUB problem. Three approaches are presented to aid in the understanding of the theoretical context of the heuristic methods examined and to highlight the formal complexity of the original problem.

Graph Theory Representation

A graphical representation of the matrix A can be constructed through the following mapping rule, f. Let each row of A be a vertex of the graph. Should two rows of A conflict then the two vertices of the graph are joined by an edge. This mapping retains all the necessary conflict information. If two vertices, a and b, are joined by an edge, e, then a and b are adjacent, and a (or b) is *incident with* e. If a and b are not adjacent, this indicates that the corresponding two rows in A do not conflict.

This introduces the notion of *independence*. Given a graph $G = (V, E)$, a subset $V' \in V$ is said to be an *independent set* if no two of its elements are adjacent. It follows that if an independent set of vertices can be found in G then the corresponding rows of the matrix A do not conflict and thus define a GUB set. Conversely, a GUB set for A defines an independent set for the graph G. It is also clear that an independent set for G is maximum if and only if the corresponding GUB set for A is maximum.

Consider the set $\mathcal{A}_m$, the set of all A-type matrices having m rows. The above mapping factors this set into a definite number of *classes*. Two matrices, A_1 and A_2 are said to belong to the same class, C, if and only if each is mapped into the same graph, G_c. Thus, an independent set of vertices of G_c correspond to a GUB row set for every matrix in the class C.

The incidence matrix N is defined with $n_{ij} = 1$ if vertex i is incident with edge j, and $n_{ij} = 0$ otherwise. There exists one, and only one incidence matrix for each graph of $\mathcal{G}$, where $\mathcal{G}$ is the set of all graphs having m vertices.

Since the set of all N-type matrices with m rows is a subset of $\mathcal{A}_m$, every class of $\mathcal{A}_m$ contains one and only one incidence matrix. In general, for the GUB problem, every m row matrix is equivalent to one of a finite number of incidence matrices. Superficially this may seem to be a simplification. But as shown in Section 7 the GUB problem on

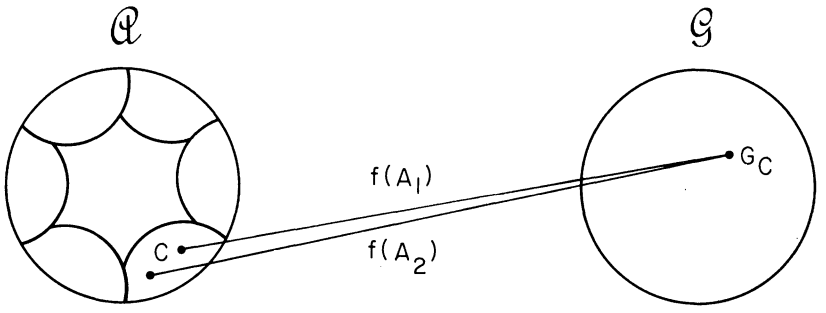

FIGURE 1.

N is as difficult as the independent set problem on G. The equivalent statements of the GUB problem do, however, offer different views of the problem which are helpful in considering algorithms for and analysis of the problem. (Note: In Garey and Johnson [6] it is shown that two other graph problems, the "vertex cover" and the "clique" problem, are equivalent to the independence problem, and hence the GUB problem. These problems do not seem to offer any additional insight for the GUB problem.)

Conflict Matrix Representation

The *conflict matrix* $\mathfrak{M}$ is defined with $m_{ij} = 1$ if row i conflicts with row j in (L), and $m_{ij} = 0$ otherwise. Note that this matrix is symmetric. The sum for any row (or column) indicates the number of other rows it is in conflict with, plus one. This sum indicates for any particular row how many other rows would be subsequently excluded from a GUB set by its addition.

The rows of a GUB structure can be rearranged to form an embedded identity matrix in $\mathfrak{M}$.

Vector Space Representation

Yet another heuristic approach can be modelled using vectors in an n-dimensional vector space, where n is the number of variables in the problem (L). Consider each row of A as a vector in this space, having unit length in those "dimensions" corresponding with its nonzero coefficients.

R, the resultant vector from the sum of all vectors of the rows of A, indicates the number of conflicts, plus one, associated with each variable of (L). A hypercube in n-space situated in the first orthant at the origin with length 1 in all positive directions denotes the feasible GUB region. Should R extend beyond this area, then the set of rows corresponding to the vectors determining R does not constitute a GUB structure.

A gradient vector can be calculated indicating the direction of the shortest distance to the *feasible region*. It can be used to determine which row to remove from the set to obtain the largest movement in the desired direction. When R falls within the feasible region, the set of rows determining R constitutes a GUB set.

3. Earlier Literature

Two papers dealing with efficient GUB finding methods are worthy of special note.

Brearley, Mitra and Williams [2] establish a very useful framework for study of methods for finding GUB structure, as well as an insightful discussion of these methods and a taxonomy for their classification.

They define three sets consisting of the rows of the technological matrix A. The first set, the *eligible set*, is made up of every row of A that is individually eligible to belong in the GUB set. The *structure set* is a subset of the eligible set and includes all those rows currently considered as members of the GUB set. The *candidate set* consists of those rows of the eligible set that are candidates for inclusion (or re-inclusion) in the GUB set. Every one of the methods examined in [2] is described in terms of manipulation of these sets.

Each method of building a GUB set employs one of two basic strategies. The *row-addition* strategy begins with an empty structure set. Then, based on a particular criterion for inclusion, rows are removed from the candidate set and either added to the structure set or dropped from further consideration. This procedure continues until the candidate set is empty. The rows in the structure set form an admissible GUB structure.

The *row-deletion* strategy takes the opposite approach and is divided into two phases. Methods of this type initially place all eligible rows in the structure set. This normally leads to an infeasible GUB set with many conflicting rows. Based upon the particular decision rules, rows are removed from the structure set and placed in the candidate set. The first phase of this strategy ends when a feasible structure is obtained.

A second phase involves examining the removed rows in the candidate set. Those that do not conflict with any of the members of the current structure set are taken from the candidate set and reincluded in the structure set. Those that do conflict are deleted from the candidate set and dropped from further consideration. The second phase ends when the candidate set is empty. At this point the rows of the structure set constitute an admissible GUB set.

Brearley, Mitra, and Williams examine over 18 different methods. These appraoches differ in the primary and secondary decision criteria for including (or removing) a row in the GUB structure set. The heuristic decision rules examined are based on the following model entities and combinations thereof: Include or remove a row based upon:

(a) the number of nonzero elements in the given row,

(b) the number of rows in conflict with the given row,

(c) the number of nonzero elements in rows that conflict with the given row,

(d) the row's relative weight obtained by the inner product of a vector representation of the row and a directional gradient.

These methods were implemented with an ALGOL program run on an ICL 4130 computer. Twelve linear programming problems ranging in size from 12 rows up to 166 rows were used for computational tests. The results show that those row-addition methods using heuristic (*d*) above "consistently performed very well" [2]. Similarly, those methods using heuristic (*b*) were found to perform nearly as well as (*d*).

McBride [15] compares the directional gradient method (d) with an approach suggested but not tested by Greenberg and Rarick [8]. The latter method uses the conflict matrix as does heurisitc (*b*). However, it focuses on finding a maximal embedded identity matrix within the conflict matrix, rather than using the conflict matrix to determine conflict counts, applying a specialization of the preassigned pivot procedure (P^3) normally used for reinversion [9]. McBride's results indicate that heuristic (*d*) is significantly faster. However, neither method consistently achieves a larger GUB set.

McBride also comments on the notion of a "good" GUB set. He finds merit in selecting a set of GUB rows that minimizes the nonzero build-up in the representation of the inverse transformation of the explicit basis during actual optimization. Results are also given for a restricted GUB set selection that gives priority to equality constraints. Since equality constraints are always binding in feasible solutions, the subset of the basis associated with binding constraints, or kernel [7], is expected to have fewer explicit nonzero elements.

Based upon the results in these papers, and on independent computational experience with automatic GUB factorization reported by Brown and Graves [3], the present research intially concentrated on those approaches utilizing the two most successful heuristics based on conflict and directional gradient (i.e. methods I.2, II.2, II.9 and II.10 of [2]).

The models studied in this report are of much larger scale and include mixed-integer problems as well as models for which prior GUB row sets have been manually specified.

4. Determination of the Eligible Set

The implementation of GUB in simplex algorithms usually admits only ± 1 as nonzero coefficients in the GUB rows. In linear programming, a column scaling can make each nonzero element in a GUB row ± 1. For variables of an integer or mixed integer programming problem, the columns of matrix A that correspond to integer variables cannot be scaled without inconvenience for other optimization functions depending upon the integrality condition. Therefore, nonzero elements in columns corresponding to integer variables will be modified by row scaling. If it is impossible to obtain the necessary ± 1 nonzero coefficients by row scaling and column scaling of columns corresponding to continuous-valued variables, the row is deemed not eligible for inclusion in a GUB set.

It is an objective of this research that the procedures examined for locating a GUB set in a linear programming problem be designed to be incorporated as an automatic, integral part of a contemporary optimization system of advanced design.

Each method is implemented as a feature of the read routine (written to accept input in the standard MPS format, as well as editing information indicating integer variables, scaling, and known prior GUB structure). Each method automatically examines the rows of the input and specifies a GUB set. The appropriate rows and columns are then scaled as necessary to obtain the proper GUB structure, and passed on to the optimizing portion of the system. (Note that the editing information places conditions that must be satisfied for any achievable GUB set.)

In determining the set of eligible rows, the following factors have to be considered.

a. Through the editing process, have some of the rows been dropped from the problem? If so, these "masked" rows are not eligible for inclusion in the GUB structure and are thus dropped from the set of eligible rows.

b. Through the editing process, have any rows been predesignated to be in the GUB structure? Large segments of the constraints can often be selected for the GUB set either visually or by recognition of a member of a convenient class of models. Any rows that conflict with these rows are not eligible for subsequent inclusion.

c. All rows designated "nonconstrained" (which include the objective function) are ineligible for inclusion in the GUB structure.

d. If there are any integer-valued variables, an additional check is performed. A row

in the GUB set must eventually be capable of being scaled to ± 1 nonzero coefficients. This is achieved, if necessary, through a combination of row and column scaling. However, with integer variables, column scaling is no longer advisable. Therefore any row with a nonzero element in integer columns that is not a $+1$ or -1, or capable of being rendered into a ± 1 in those positions through row scaling alone, must be marked as ineligible for inclusion in the GUB structure.

Once the above restrictions have been considered, the resulting set of eligible rows is then available for search in order to construct the desired GUB structure.

5. Implementation of Automatic GUB Heuristics

Conflict Methods

These employ the notion of a conflict measure for each row. Consider the conflict matrix, M, of the corresponding technological matrix A, for which a GUB set is to be found. An individual element, m_{ik} is 1 if row i and row k of the original matrix have at least one column j such that $a_{ij} \neq 0$ and $a_{kj} \neq 0$. If the two rows have no nonzero coefficients in a common column then the corresponding m_{ik} of the conflict matrix is 0. Summing across a row of the conflict matrix can thus give the measure of the number of rows plus one that are in conflict with a given row. For a given row, this sum less one, called the row's *deletion potential*, indicates exactly how many other rows would be immediately excluded from the GUB set by inclusion of this row.

Conflict row-addition places all the eligible rows on a candidate list. From the candidate list, individual rows are selected by *minimum* deletion potential and removed to be added to the structure set. Other rows that are in conflict with the selected row are immediately removed from the candidate list and discarded. The selection of rows for the structure set and the discarding of conflicting rows continues until the candidate list is exhausted. The resulting structure set forms a GUB set.

A modification to the above heuristic is possible which *breaks ties* among rows sharing the minimum deletion potential by (for instance) selecting the row having the most nonzero elements for inclusion with the GUB structure set.

The program used to test this heuristic approach is adapted from an earlier version made available by Graves.

Conflict Row-Addition

Step 1. Identify Eligible Rows. Set $\beta_i = 1$ if row i is an eligible row, and equal to 0 otherwise.

Step 2. Determine Deletion Potential. Scan each eligible row i and increment β_i by the number of other eligible rows k where a_{ij} and a_{kj} are both nonzero for at least one column j. (β_i is the deletion potential, plus one.)

Step 3. Stopping Condition. If any β_i is greater than 0, go to the next step. Otherwise, stop. At termination, the structure set is a GUB row set.

Step 4. Row Selection. Select row i having the minimum positive ("deletion potential") β_i and add it to the structure set.

Step 5. Exclude Rows in Conflict with Selected Row. Locate the $(\beta_i - 1)$ rows in conflict with the selected row. For each of these rows k, locate the $(\beta_k - 1)$ rows that they are in conflict with and decrement β_i for those rows by one.

Step 6. Marking Selected and Excluded Rows Ineligible for Further Consideration. Set β_i and the β_k's equal to zero. Go to step 3.

Conflict Row-deletion is a two-phase method which initially places all the eligible rows in the structure set. From this set individual rows are selected during Phase 1 and placed on the candidate list by *maximum* deletion potential. During Phase 2, remaining candidate rows that do not conflict with the structure set can be reconsidered in LOFI order [2]. A modification of phase 2 is used in this research which simply excludes from further consideration all conflicting rows, reincludes any remaining candidate rows, and repeats phase 1, until no further nonconflicting candidates remain.

Gradient Methods

Gradient row-deletion employs a heuristic method suggested by Senju and Toyoda [17] for approximate solution of certain linear programming problems with 0, 1 variables. The objective is to obtain a maximum number of rows in the GUB structure while satisfying the stipulation that the GUB rows be disjoint.

$$
\begin{aligned}
&\text{Max } Z \quad = x_1 + x_2 + \ldots + x_m \\
&\text{s.t.} \qquad \sum_{i\,:\,a_{ij}\neq 0} x_i \leqslant 1, j = 1, \ldots, n \\
&\text{where} \quad x_i \in \{0, 1\}, \\
&\qquad m \text{ is the number of candidate rows in } (L), \\
&\qquad n \text{ is the number of variables in } (L), \\
&\qquad x_i \text{ is the variable which detemines whether row } \boldsymbol{i} \text{ is in the} \\
&\qquad\quad \text{GUB set or not, and} \\
&\qquad Z \text{ is the objective function.}
\end{aligned}
\tag{S}
$$

Using the vector space viewpoint outlined earlier, consider each row of (S) as a vector in n-space. A resultant vector R is determined by the sum of all the included rows and, in general, extends beyond the feasible space denoted by the unit hypercube. A gradient vector is calculated from this infeasible point in the direction of the shortest distance to the feasible region. An inner product of this gradient with each of the row vectors results in a relative weight for each row, which can be viewed as indicating the relative contribution that removal of the row would have towards obtaining a feasible structure set.

Rows are removed from the structure set according to their relative weight, the largest weight being removed first. This process is continued until a feasible set of GUB rows has been obtained. (The gradient vector is not recomputed as the method proceeds.)

Next, a phase 2 procedure examines each of the initially removed rows to see if any can be reincluded into the structure set without violating the GUB restrictions. Upon completion of phase 2, the selected rows constitute a GUB set.

A variation on the above procedure recalculates the shortest distance to the feasible region after the removal of each row. With the new gradient, a new set of relative weights for the remaining rows is then calculated and used, if necessary, to determine which of the subsequent rows will be removed.

Another modification is possible whenever two rows are found with equal weights. As a tie-breaking rule, the row found to have the least number of nonzero coefficients may be discarded first.

Gradient Row-Deletion

Phase 1: Deletion of Infeasible Rows

Step 0. Initialize Sets. Add all eligible rows to the structure set. The candidate set is empty.

Step 1. Determine the Vector R. For each column j, define ρ_j as the number of rows in the structure set having nonzero elements in column j.

Step 2. Determine Relative Weight of Each Row. For each row i, define v_i as the sum of $(\rho_j - 1)$ of every column j, for which $a_{ij} \neq 0$.

Step 3. Feasibility Condition. If for every column, $\rho_j \leqslant 1$, then go to step 6; else find a column j such that $\rho_j > 1$.

Step 4. Determine Row for Exclusion. Examine the rows in the structure having nonzero elements in column j. Select the row i with the largest v_i.

Step 5. Remove Selected Row. Remove row i from the structure set, decrementing ρ_j by one for every column j with $a_{ij} \neq 0$. Add row i to the candidate set and return to step 3.

Phase 2: Improve Feasible GUB Set Found by Re-including Excluded Rows

Step 6. Eliminate Rows in Candidate Set that Conflict with the Feasible Set. For every row i of the candidate set that has at least one $a_{ij} \neq 0$ in a column with $\rho_j = 1$, remove that row from the candidate set.

Step 7. Re-inclusion of Rows. If any rows remain in the candidate set, then find row i having the smallest v_i. Remove row i from the candidate set and re-include it in the structure set. Increment ρ_j by one for every column j where $a_{ij} \neq 0$.

Step 8. Stopping Condition. If the candidate set is empty, stop; else go to step 6.

To modify the algorithm in order to compute a new gradient vector after the removal of each row in phase 1, step 5 is changed as follows:

Step 5*. Remove Selected Row. Remove row i from the structure, decrementing ρ_j by one for every column j such that $a_{ij} \neq 0$. Locate each row k that is in conflict with row i. Decrement v_k by the number of conflicts between the two rows. Add row i to the candidate set and return to step 3.

These two basic methods have been implemented as integral modules of a large scale optimization system. Therefore, explicit conflict matrices are not built. (To have done so would have consumed too much computer time and space.) Instead, all the information is stored in the vectors β, ρ, and v. Logical flags associated with each row indicate whether it is eligible, and whether it is in the candidate set or in the structure set.

The problem data is expressed internally in terms of only the unique nonzero elements. This input is stored in a doubly linked list having both a row and a column thread. Thus, along with any nonzero coefficient a_{ij}, the location of adjacent nonzero elements in both the row i and column j are also immediately available. This crucial feature permits efficient row access for various operations (e.g., to locate all rows that conflict with a given row at a particular column).

6. Computational Results

The heuristic methods have been tested on 15 problems that vary in size from 92 constraints to 4,648 constraints. A description of each of the problems is given in Figure 2. As can be seen, four of the problems are mixed integer and two are pure integer.

Problem	Number of rows	Number of columns	Integer Columns	Nonzeros
VANN	92	1,324	1,324	2,648
NETTING	103	247	103	494
AIRLP	171	3,040	0	6,023
COAL	171	3,753	0	7,506
TRUCK	239	4,752	4,752	30,074
CUPS	415	619	145	1,341
FERT	606	9,024	0	40,484
PIES	663	2,923	0	13,288
PAD	695	2,934	0	13,459
ELEC	785	2,800	0	8,462
GAS	799	5,536	0	27,474
FOAM	1,017	4,020	42	17,187
LANG	1,236	1,425	0	22,028
JCAP	2,487	3,849	560	9,510
ODSAS	4,648	4,683	0	30,520

FIGURE 2.

The results of these experiments are given in Appendix A. The first two columns give the rows and nonzero column elements, respectively, of the GUB structures found. The time given in column three is the time required to locate the GUB set once the set of eligible rows has been determined. The final columns give additional information relating to the two versions of the gradient methods examined and represents total time in phase 1 and the number of rows reincluded in the GUB structure during phase 2.

As with the earlier work cited, the Senju and Toyoda methods were found to be consistently the faster. Gradient row-deletion which updates the gradient after each row is removed takes longer in phase 1 than its nonupdating counterpart. However, it so selectively deletes the rows, that few if any rows are ever added back into the structure during phase 2. This suggests that it be implemented as strictly a one-phase method.

All methods are robust in that they consistently find large GUB sets. The conflict approaches generally find a larger number of variables with nonzero coefficients in the GUB rows. However, they definitely become relatively inefficient when larger problems are analyzed, regardless of the relative size of the GUB structure in the problem.

There is some discrepancy between these results and those published earlier [2]. The wide variation between gradient row-deletion with, and without, gradient updating has not been observed in the current experiments. It is hypothesized that this is due partially to differences in implementation of the various approaches and partially to problem size and structure variations between these studies.

7. Problem Complexity

The *complexity* of a problem is said to be polynomial if an algorithm exists for which the fundamental operations are limited by a polynomial function of intrinsic problem dimensions. Such an algorithm would be called a *polynomial time* or *good* algorithm. The class of all problems for which such algorithms exists is denoted (P). If an algorithm is not polynomial time, then it is defined to be an *exponential time* algorithm.

The disadvantage of an exponential algorithm is the explosive growth of the maximum solution time as the dimensions of the problem increase [14].

A problem x is said to be *reducible* to a problem y if each *good* algorithm for solving y can be used to produce in polynomial time a good algorithm for solving x [12]. Note that this does not necessarily require that a good algorithm for x and y actually exist. This requires only that if one exists for y, then one also exists for x.

An *intractable* problem is one for which it is known that no polynomial time algorithm exists. In between this class of problem, and the class P, is a vast number of problems whose status is uncertain. Among these is a class of *nondeterministic polynomial-time* problems (NP) for which a polynomial-time algorithm can be shown to exist that can *verify* a guessed solution, but for which the existence of a (deterministic) polynomial-time algorithm to actually solve a problem has not yet been demonstrated.

If every problem of the class NP is reducible to the problem y, then y is said to be NP-*hard*. In addition, if y itself belongs to NP, then y is NP-*complete* [6], [12].

The following problem is known as the *independent set decision problem* (ISD). It belongs to the set of NP-complete problems.

(ISD) Given a graph $G = (V, E)$ and an integer t, decide whether G contains an independent set of size t or more.

The GUB decision problem (GUBD) can be defined as follows:

(GUBD) Given an integer p and an $m \times n$ matrix K defined as $K_{ij} = 1$ if $a_{ij} \neq 0$, and $K_{ij} = 0$ otherwise, decide whether K contains a set of p or more rows $i_1, i_2, \ldots, i_q$ such that

$$\sum_{e=1}^{q} k_{i_e j} \leqslant 1 \quad \text{for every column; } q \geqslant p. \tag{*}$$

Given an instance of the ISD problem, the incidence matrix N can be constructed. This matrix along with the integer t is an instance of the GUBD problem. The following theorem proves the correctness of this reduction:

THEOREM. *The incidence matrix N has t rows satisfying (*) if and only if there are t vertices in G that are independent.*

PROOF. a) Assume there exists t rows of N that satisfy (*). They correspond to vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_t}$ in G. If any two of these vertices are adjacent, then

$$\sum_{e=1}^{t} n_{i_e j} = 2$$

where j is the column in N that corresponds to the edge connecting the two vertices. This is a violation of the assumption, hence the t vertices in G are not connected to one another.

b) Assume there exists t vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_t}$ in G that are independent. Since no two are adjacent, the corresponding rows in N satisfy (*) [19]. Q.E.D.

Since the ISD problem, a problem known to be NP-complete, is reducible to the GUBD problem, it follows that the GUBD problem itself is NP-complete. (It is clear that the reduction is polynomial time and it is also clear that GUBD is in NP.) The related problems of finding a maximum independent set and a maximum GUB set are not in NP, however, they are NP-hard. It is therefore unlikely that a polynomial-time

algorithm will be found for these problems. Only exponential-time algorithms are presently available.

The above analysis of GUB algorithms has only addressed the *worst case* bound. No conclusions are made about the average performance of an algorithm. In other words, the possibility of the existence of an algorithm with good average performance, but having an exponential worst case bound, has not been ruled out.

8. Upper Bounds for the Size of Maximum GUB Set

The intrinsic difficulty of identifying a maximum GUB set has been shown to be essentially impossible for problems of the scale at hand. However, the efficient heuristic procedures have been shown to provide very large GUB sets, whose size appears to be relatively stable for each problem regardless of the particular method applied. This suggests that these large GUB sets may be, in fact, very nearly maximum, although there is no practical way to verify this directly.

Although the problem of determining the size of the maximum GUB set is also NP-hard, it is possible to develop an easily computable *upper bound* on the maximum GUB set size. This bound can then be used to objectively evaluate the quality of the GUB sets produced by heuristic algorithms.

It is clear that the number of rows of a GUB set can be no greater than the number of rows in the problem. Also any one row by itself can form a GUB set. But these bounds are of little practical use when considering the problem of identifying a maximum GUB set. Utilizing information that is already available in the heuristic procedure, it is possible to construct in polynomial time an upper bound on the size of the maximum GUB set. (It is also possible to construct a lower bound on the size of the maximum GUB set, but that topic is not pursued in this report.)

For the purpose of developing a better bound, the incidence matrix representation (N) of the problem is used. Let s_i be the number of 1's in row i. Note that s_i is the number of edges incident to vertex i in G. Also note that $s_i = \beta_i - 1$. The number of columns in N represents the number of distinct conflicts that exist between the rows of the original problem. This number is denoted as c, and can be found by the following formula

$$c = \frac{\sum_{i=1}^{n} s_i}{2} .$$

If c is greater than 0, all the rows of N cannot simultaneously belong to a GUB set, which implies the cardinality of the GUB set is less than m. As c becomes larger, the following argument shows that the upper bound of the maximum GUB set decreases.

If c is positive, but strictly less than m, it is possible for all the conflicts to involve one row. Removal of that row would then leave $m - 1$ rows that form a GUB set. Thus for c in the range from 1 to $m - 1$, an upper bound on the size of the maximum GUB set is $m - 1$. Since one row can conflict with at most $m - 1$ other rows, once $c \geqslant m$, at least two rows have to be removed to form a GUB set. For $m < c < [(m - 1) + (m - 2)]$ it is possible to construct an incidence matrix such that all the conflicts are between a pair of rows and the remaining set of rows. Removal of the pair would result in a GUB set of $m - 2$ rows. This constructive argument continues until $c = [(m)(m - 1)]/2$, which occurs when each row conflicts with every other row. At that point, the max maximum GUB = min maximum GUB = one row.

In general, for any problem with an $m \times c$ incidence matrix, the largest maximum GUB set that can be obtained is:

$$u_1 = \llcorner 0.5 + \sqrt{0.25 + (m)(m-1) - 2c}$$

where $\llcorner$ indicates truncation to an integer.

The above bound is *problem-independent* and a *sharp* bound in that matrices with a GUB set the size of the bounding value can be constructed.

With additional information about a specific problem a better bound can be constructed. Since s_i is the number of other rows that conflict with row i, removing row i from the set of rows reduces the number of conflicts, c, by s_i. Let y denote max s_i. Since y is the largest row conflict count, c can be reduced by not more than y with the removal of each row. The minimum number of rows that would have to be removed to reduce the number of row conflicts to 0, is $\ulcorner c/y$. Therefore, given m, c and y, the bound can be improved to

$$u_2 = \begin{cases} m - \ulcorner \dfrac{c}{y}, & c \leqslant (m-y)y, \\ \llcorner 0.5 + \sqrt{0.25 + y(2m - y - 1) - 2c}, & c > (m-y)y; \end{cases}$$

where $\ulcorner$ indicates rounding up to the next integer.

In order to determine y, the entire β vector must be examined.

A third, even better bound can be obtained with additional information on the *frequency* of the conflict counts from 1 to y. The procedure is the same as above, in that when a row is removed with y conflict count, c decreases by y. However, instead of continuing to decrease c by y; it is decreased by the next largest s_i. This procedure continues until, once again, c becomes zero. This bound is named u_3.

The bounds developed can be used to objectively evaluate the size of a GUB set found by heuristic methods. In two problems examined, VANN and AIRLP, the number of rows in the GUB set equal an upper bound on the maximum GUB set for the problem. Therefore, for those problems, the heuristic methods are verified to have located maximum GUB sets.

Manual specification of a GUB set from visual inspection can utilize these bounds as an excellent measure of the maximum additional rows to be found. This information is also an aid in deciding whether to subject the problem to additional automatic search for GUB.

9. Extensions

The upper bounds developed in this report vary from a problem-independent bound to tighter problem-dependent bounds. It is speculated that additional information can be easily extracted from the actual conflict structure of the problems that can be used to tighten the existing bounds even further. This is strongly suggested by manual analysis of problems with particularly loose bounds for which the conflict structure seems to have higher-order pathology. In addition, lower bounds have been developed by similar methods.

Another area that warrants further study is the special structure of the incidence matrix representation of the original problem. It is noted that for an incidence matrix, N, the relative weights generated for each row are (except for a constant) identical for

both the conflict and the gradient methods studied. This implies that for a matrix N, the row-deletion heuristics will identify the same GUB set.

As things now stand, GUB-finding demands far less cost than the benefits derived during model optimization. Better GUB-finding methods may result from simple extensions arising from relaxations of (S), use of conflict information of higher order, limited application of backtracking enumeration, or exploitation of conditioned bounds on the remaining candidate rows to allocate heuristic effort.

Finally, research is continuing on automatic location of network row structure (e.g., Musalem [16] and Wright [18]). As one illustration of an immediate generalization of the GUB results, a GUB set for a problem can be identified and then another GUB set of an eligible subset of remaining rows can be found. Thus, a *bi-partite network row factorization* can be achieved (e.g., transportation or assignment rows).

10. Conclusions

The computational benefits of a large GUB set for an LP problem are widely recognized. This report shows that the identification of a maximum GUB set is a difficult problem, essentially as hard as many other widely known difficult problems.

The use of heuristics seems inescapable. This report has examined two promising heuristics (with two versions of each) applied to a series of real-life, large-scale models. All versions are robust in their ability to find large GUB row sets. However the two versions that use the Senju and Toyoda method are consistently the fastest. These two methods are essentially equal in their efficiency and effectiveness. Since the version which recalculates the gradient after the removal of each row so selectively removes the rows during the first phase that few if any rows are re-included in the GUB set during the second phase, this suggests that the latter phase be omitted.

The representation of an infinite number of m-row matrices by a finite number of incidence matrices offers a powerful and concise way of examining the GUB problem. Under this representation, both basic heuristic methods investigated assign (within a constant) the same relative selection weights to each row.

Finally, the ability to define upper bounds on the maximum size of the GUB set gives a new powerful tool in this area. It enables one to evaluate the quality of GUB sets found even in every large problems, for which the algorithmic identification of a maximum GUB set is probably impossible in general. In some cases, verification of a heuristically achieved maximum GUB set is now possible. Further, the bounds developed may be further enhanced in future research, and may be applicable to related problems of equivalent complexity.[1]

Appendix A

This appendix contains computational results for fifteen linear, mixed-integer and integer models. All execution times reported are expressed in actual CPU seconds, accurate to the precision displayed for IBM 360/67 and FORTRAN H (Extended).

For clarity, the following terms are defined:

Eligible rows: The number of rows of the model initially eligible for inclusion in a set of GUB rows.

[1]The authors wish to thank Gordon Bradley and Shmuel Zaks for their insights on complexity, and also Glenn Graves and William Wright for thier considerable assistance.

Conflict count: The number of columns of the incidence matrix for the problem.
Conflict density: The ratio of the conflict count to the maximum conflict count for that problem size [i.e., $m(m-1)/2$].
Time to find Elig: The time in CPU seconds to determine the set of eligible rows.
IMAX: The maximum of s_i.
U_1, U_2, U_3: Bounds defined in Section 8.

The methods are labelled:

CRA Conflict Row-Addition
CRD Conflict Row-Deletion
GRD* Gradient Row-Deletion (with gradient update)
GRD Gradient Row-Deletion

Problem:	VANN	Description:	Fleet Dispatch Model		
Rows:	92	Eligible rows:	69	IMAX:	0
Columns:	1324	Conflict count:	0	U1:	69
Integer:	1324	Conflict density:	0	U2:	69
Nonzero:	2648	Time to find Elig:	0.141 sec	U3:	69

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	69	1324	0.237		
CRD	69	1324	0.125		
GRD*	69	1324	0.202	0.198	0
GRD	69	1324	0.202	0.198	0

Problem:	NETTING	Description:	Currency Exchange Model		
Rows:	103	Eligible rows:	71	IMAX:	5
Columns:	247	Conflict count:	46	U1:	70
Integer:	103	Conflict density:	1.85%	U2:	59
Nonzero:	494	Time to find Elig:	0.022 sec	U3:	46

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	36	84	0.169		
CRD	36	84	0.164		
GRD*	36	i7	0.047	0.042	0
GRD	36	72	0.042	0.037	0

Problem:	AIRLP	Description:	Fleet Dispatch Model		
Rows:	171	Eligible rows:	170	IMAX:	150
Columns:	3040	Conflict count:	2983	U1:	151
Integer:	0	Conflict density:	20.77%	U2:	150
Nonzero:	6023	Time to find Elig:	0.076 sec	U3:	150

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	150	3000	1.16		
CRD	150	3000	0.761		
GRD*	150	3000	0.645	0.639	0
GRD	150	3000	0.444	0.439	0

Problem:	COAL	Description:	Energy Development Model		
Rows:	171	Eligible rows:	170	IMAX:	111
Columns:	3753	Conflict count:	3753	U1:	146
Integer:	0	Conflict density:	26.13%	U2:	136
Nonzero:	7506	Time to find Elig:	0.106 sec	U3:	121

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	111	3753	1.38		
CRD	111	3753	1.24		
GRD*	111	3753	0.920	0.912	0
GRD	100	2568	0.641	0.631	0

Problem:	TRUCK	Desciption:	Fleet Dispatch Model		
Rows:	239	Eligible rows:	221	IMAX:	171
Columns:	4752	Conflict count:	10438	U1:	165
Integer:	4752	Conflict density:	42.94%	U2:	159
Nonzero	30074	Time to find Elig:	0.116 sec	U3:	144

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	32	1069	6.88		
CRD	30	1099	7.095		
GRD*	30	857	5.00	4.95	2
GRD	32	986	1.70	1.58	8

Problem:	CUPS	Description:	Production Scheduling Model		
Rows:	415	Eligible rows:	390	IMAX:	48
Columns:	619	Conflict count:	744	U1:	388
Integer:	145	Conflict density:	0.98%	U2:	374
Nonzero	1341	Time to find Elig:	0.042 sec	U3:	294

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	213	494	2.96		
CRD	214	442	3.15		
GRD*	214	466	0.212	0.194	0
GRD	200	394	0.384	0.132	24

Problem:	FERT	Description:	Production & Distribution Model		
Rows:	606	Eligible rows:	605	IMAX:	580
Columns:	9024	Conflict count:	16455	U1:	577
Integer:	0	Conflict density:	9.01%	U2:	576
Nonzero:	40484	Time to find Elig:	0.257 sec	U3:	567

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	559	9024	15.8		
CRD	559	9024	10.5		
GRD*	559	9024	6.73	6.71	0
GRD	559	9024	2.52	2.50	0

Problem:	PIES	Description:	Energy Production & Consumption Model		
Rows:	663	Eligible rows:	662	IMAX:	21
Columns:	2923	Conflict count:	4116	U1:	655
Integer:	0	Conflict density:	1.88%	U2:	466
Nonzero:	13288	Time to find Elig:	0.866 sec	U3:	422

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	180	1848	10.8		
CRD	169	1693	13.5		
GRD*	172	1811	2.82	2.77	1
GRD	177	1761	1.31	0.788	28

Problem:	PAD	Description:	Energy Production & Consumption Model		
Rows:	695	Eligible rows:	694	IMAX:	23
Columns:	2934	Conflict count:	4416	U1:	687
Integer:	0	Conflict density:	1.84%	U2:	502
Nonzero:	13459	Time to find Elig:	0.104 sec	U3:	449

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	200	1864	13.1		
CRD	189	1771	16.6		
GRD*	188	1708	3.34	3.26	2
GRD	189	1275	1.35	0.928	21

Problem:	ELEC	Description:	Energy Production & Consumption Model		
Rows:	785	Eligible rows:	784	IMAX:	22
Columns:	2800	Conflict count:	6167	U1:	776
Integer:	0	Conflict density:	2.01%	U2:	503
Nonzero:	8462	Time to find Elig:	0.089sec	U3:	492

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	309	2461	11.4		
CRD	210	2791	16.1		
GRD*	309	2641	1.15	1.12	0
GRD	309	2605	0.842	0.579	14

Problem:	GAS	Description:	Production Scheduling Model		
Rows:	799	Eligible rows:	789	IMAX:	608
Columns:	5536	Conflict count:	22220	U1:	760
Integer:	0	Conflict density:	7.15%	U2:	752
Nonzero:	27474	Time to find Elig:	0.151 sec	U3:	652

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added In Phase 2
CRA	583	5102	16.2		
CRD	639	5536	10.4		
GRD*	608	5309	3.79	3.77	0
GRD	639	5533	1.47	1.44	1

Problem:	FOAM	Description:	Production Scheduling Model		
Rows:	1017	Eligible rows:	1006	IMAX:	261
Columns:	4020	Conflict count:	8186	U1:	997
Integer:	42	Conflict density:	1.62%	U2:	974
Nonzero	17187	Time to find Elig:	0.198 sec	U3:	934

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	932	4020	23.4		
CRD	932	4020	9.47		
GRD*	917	3981	1.73	1.71	0
GRD	917	3981	0.902	0.879	0

Problem:	LANG	Description:	Equipment &Manpower Scheduling Model		
Rows:	1236	Eligible rows:	1235	IMAX:	184
Columns:	1425	Conflict count:	46424	U1:	1196
Integer:	0	Conflict density:	6.09%	U2:	982
Nonzero:	22028	Time to find Elig:	0.072 sec	U3:	973

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	382	1207	46.2		
CRD	338	908	54.2		
GRD*	342	923	14.9	14.8	2
GRD	342	922	12.4	1.13	234

Problem:	JCAP	Description:	Production Scheduling Model		
Rows:	2487	Eligible rows:	2446	IMAX:	488
Columns:	3849	Conflict count:	16578	U1:	2439
Integer:	560	Conflict density:	0.55%	U2:	2412
Nonzero:	9510	Time to find Elig:	0.265 sec	U3:	1812

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	529	2072	104		
CRD	512	2186	153		
GRD*	529	2087	2.23	1.87	5
GRD	523	1393	3.98	1.10	59

Problem:	ODSAS	Description:	Manpower Planning Model		
Rows:	4648	Eligible rows:	4647	IMAX:	4194
Columns:	4683	Conflict count:	5220	U1:	4645
Integer:	0	Conflict density:	0.05%	U2:	4645
Nonzero:	30520	Time to find Elig:	0.263 sec	U3:	4024

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	751	3116	369		
CRD	721	3846	651		
GRD*	749	4436	7.12	6.88	0
GRD	751	3020	3.01	2.57	2

References

1. BEALE, E. M. L., "Advanced Algorithmic Features for General Mathematical Programming Systems," *Integer and Nonlinear Programming*, ed. J. Abadie, North-Holland/American Elsevier New York, 1970, pp. 119–137.
2. BREARLEY, A. L., MITRA, G. AND WILLIAMS, H. P., "Analysis of Mathematical Programming Problems Prior to Applying the Simplex Algorithm," *Math. Programming*, Vol. 8 (1975), pp. 54–83.
3. BROWN, G. AND GRAVES, G., "Design and Implementation of a Large Scale (Mixed Integer, Nonlinear) Optimization System," paper presented at ORSA/TIMS Las Vegas, Nov. 1975.
4. ——— AND Thomen, D., "Automatic Factorization of Generalized Upper Bounds in Large-Scale Optimization Problems," NPS55-80-003, Naval Postgraduate School (January 1980).
5. DANTZIG, G. B. AND VAN SLYKE, R. M., "Generalized Upper Bounding Techniques," *J. Comput. and System Sci.* Vol. 1 (1967), pp. 213–226.
6. GAREY, M. R. AND JOHNSON, D. S., *Computers and Intractability; A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, Calif., 1979.
7. GRAVES, G. W. AND MCBRIDE, R. D., "The Factorization Approach to Large-Scale Linear Programming," *Math. Programming*, Vol. 10 (1976), pp. 91–110.
8. GREENBERG, H. J. AND RARICK, D. C., "Determining GUB Sets via an Invert Agenda Algorithm," *Math. Programming*, Vol. 7 (1974), pp. 240–244.
9. HELLERMAN, E. AND RARICK, D., "Reinversion with the Preassigned Pivot Procedure," *Math. Programming*, Vol. 1 (1971), pp. 195–216.
10. HIRSHFELD, D. S., "Generalized Upper Bounding (GUB): Theory, Applications and Performance," paper presented at Share XXXV, Mathematical Programming Project, August 1970.
11. KAUL, R. N., "An Extension of Generalized Upper Bounded Techniques for Linear Programming," (ORC65-27) Operations Research Center, University of California, Berkeley, 1965.
12. KLEE, V., "Combinatorial Optimization: What is the State of the Art?" paper presented at SIAM Applied Mathematics Conference, Monterey, CA, February 1978.
13. LASDON, L. S., *Optimization Theory for Large Systems*, MacMillan, New York, 1970.
14. LEWIS, H. R. AND PAPADIMITRIOU, C. H., "The Efficiency of Algorithms," *Scientific Amer.*, Vol. 238, No. 1 (1978), pp. 96–109.
15. McBride, R. D., "Linear Programming with Linked Lists and Automatic Guberization," Working Paper No. 8175, University of Southern California, School of Business, July 1975.
16. MUSALEM, J. S., "Converting Linear Models to Network Models," Ph.D. Dissertation, UCLA (December 1979).
17. SENJU, S. AND TOYODA, Y., "An Approach to Linear Programming with 0-1 Variables," *Management Sci.*, Vol. 15 (1968), pp. B196–B207.
18. WRIGHT, W., "Embedded Network Identification in Large Scale Optimization," M. S. Thesis, Naval Postgraduate School, June 1980.
19. ZAKS, S., Private Communication, September 1979.