1984

Automatic identification of embedded network rows in large-scale optimization models

Brown, Gerald G.; Wright, William G.

Springer

Brown, Gerald G., and William G. Wright. "Automatic identification of embedded network rows in large-scale optimization models." Mathematical Programming 29.1 (1984): 41-56.

http://hdl.handle.net/10945/63200

Mathematical Programming 29 (1984) 41–56
North-Holland

AUTOMATIC IDENTIFICATION OF EMBEDDED NETWORK ROWS IN LARGE-SCALE OPTIMIZATION MODELS

Gerald G. BROWN and William G. WRIGHT
Naval Postgraduate School, Monterey, CA 93940, USA

Received 1 December 1980
Revised manuscript received 17 May 1983

The solution of a large-scale linear, integer, or mixed integer programming problem is often facilitated by the exploitation of special structure in the model. This paper presents heuristic algorithms for identifying embedded network rows within the coefficient matrix of such models. The problem of identifying a maximum-size embedded pure network is shown to be among the class of NP-hard problems. The polynomially-bounded, efficient algorithms presented here do not guarantee network sets of maximum size. However, upper bounds on the size of the maximum network set are developed and used to show that our algorithms identify embedded networks of close to maximum size. Computational tests with large-scale, real-world models are presented.

Key words: Networks, Large-Scale Optimization, Basis Factorization, Computational Complexity, Mixed Integer Optimization, Generalized Upper Bounds.

1. Introduction

The success of mathematical optimization and the increase in size and speed of digital computers have led to the modeling of very large and complex systems as mathematical programs. The direct solution of the associated linear programming (LP) problems using the classical simplex method is often expensive, if not practically impossible.

Large-scale models frequently have sparse coefficient matrices with special structure. If special structure can be identified, it can often be used to reduce the apparent problem monolith to components of more manageable size, or to admit enhancement of solution procedures. Possible ways of exploiting the structure are either factorization algorithms, for which all simplex bases share a common structure under row partition, or decomposition. The details of actual exploitation of special structure, once identified, will not be discussed here (e.g., see [8] or [9]).

Useful factorizations (even for a subset of columns) include simple bounds, generalized (upper) bounds (GUB), and embedded network rows, among others. Simple bound rows have only one non-zero coefficient. GUB refers to a set of rows for which each column has at most one non-zero coefficient in the set. Embedded generalized network (GN) rows refers to a set of rows for which each column has at most two non-zero coefficients in the set. If the nonzero coefficients in the embedded network rows are restricted to at most one +1 and one −1 in each column, then the structure is referred to as an embedded pure network (NET).

Various methods are available to identify special structure in the coefficient matrix. These range from simple permutation of rows and columns to full (linear) transformations of the coefficient matrix. An intermediate method allows simple scaling (multiplication by a nonzero constant) of each row and/or column. Generally, entire transformation methods are used in an attempt to convert the complete coefficient matrix to one having a very special structure, such as a node-arc incidence matrix for a network. Partial transformation methods look for large subsets of the coefficient matrix which exhibit the desired structure, with the implicit presumption that large subsets are more efficiently exploited than small subsets.

Much of the computational improvement of the specialized simplex algorithms is obtained when logic can be substituted for arithmetic in simplex operations. This is most conveniently accomplished when the coefficient values in the special structure set are restricted to $0, \pm 1$. It is often possible, through row and/or column scaling, to create additional ± 1 values. For simple upper bounds, row scaling will suffice. GUB sets can be converted with row and column scaling (except that columns corresponding to integer variables are not customarily scaled). To produce pure network rows, however, the scaling problem is nontrivial due to the existence of two nonzero coefficients in many columns as well as the requirement that unit elements in the same column be of opposite sign.

The use of GUB has received much attention since the concept was introduced in 1964 by Dantzig and Van Slyke [6]. Some form of GUB has been implemented in many commercial LP systems, though restrictions on what constitutes an admissible (i.e. implemented) GUB set vary. Work has been done in the automatic identification of GUB sets [2, 3]; computational results on large-scale problems indicate that this is not only feasible, but can be extremely advantageous [3, 17].

Although some elegant work has been done in the theory of entire conversion of a linear program to a pure network problem [1, 14], few practical results have been achieved which reliably identify a subset (of rows) which forms a network structure if entire conversion fails. An efficient algorithm for doing so is of considerable value because a model usually fails to be completely convertible, and because the expense of attempting entire conversion may be prohibitive.

The problem of finding a maximum GUB set (in terms of number of rows) within a general coefficient matrix has been shown by Thomen to be NP-hard [17]. We prove the same result for the maximum embedded pure network problem. The implication is that currently only exponential-time algorithms exist to solve these identification problems and the hope of finding a more efficient algorithm is dim.

Therefore, the efficient identification methods we have developed have been heuristic algorithms. They find large, sometimes even maximum structures, but they cannot guarantee a maximum result. Since the size of the maximum structure is not known for the large-scale problems with which we work, we develop upper bounds on this size to evaluate our heuristics [17].

Computational results are given for a number of large-scale, real-world problems. They show the NET identification algorithms to be very effective and efficient in

identifying large sets of pure network rows. ([4] presents equivalent results for GN identification.)

Some of this research has been summarized in [5].

2. Problem definition and representations

The Linear Porgramming Problem is defined here as:

$$\begin{aligned} (\mathrm{L}) \quad & \text{minimize} \quad cx \\ & \text{s.t.} \quad \underline{r} \leq Ax \leq \bar{r} \quad \text{(ranged constraints)}, \\ & \qquad \underline{b} \leq x \leq \bar{b} \quad \text{(simple bounds)}, \end{aligned}$$

where $\underline{r}$ and $\bar{r}$ and m-vectors, x, c, $\underline{b}$ and $\bar{b}$ are n-vectors and A is an $m \times n$ matrix. Consider for the moment the case where all variables, x, are real-valued; the integer and mixed integer cases are admitted later.

The (maximum) GUB problem for (L) can be stated as:

(GUB) Find a (maximum) subset of rows in A which can be scaled to contain only $0, +1$ entries and which satisfy the property that each column of A has at most one nonzero entry in the subset.

The real values of the nonzero coefficients in A do not make a difference in the GUB problem, because any nonzero entry in a GUB row can be scaled to $+1$ by column scaling alone. Therefore, it is convenient to replace A by a binary $(0, 1)$ matrix, K, of the same dimension where each nonzero entry of A is replaced by $+1$ with all other entries zero.

Using the matrix K, with entries k_{ij}, the (maximum) GUB problem can be formulated as the binary integer program

$$\begin{aligned} (\mathrm{GUBI}) \quad & \text{(maximize)} \quad z_1 + z_2 + \cdots + z_m \\ & \text{s.t.} \quad \sum_i k_{ij} z_i \leq 1, \quad j = 1, \ldots, n \\ & \qquad \text{where} \quad z_i \in \{0, 1\}. \end{aligned}$$

(z_i is an indicator variable for GUB inclusion.)

Alternate representations of the GUB problem have been developed as the basis for various heuristic algorithms and for theoretical considerations such as determining the complexity of the problem and developing bounds on the maximum achievable size of a GUB set. These include graphical conflict matrix, conflict matrix, and vector space representations [17].

Two rows in A are said to *conflict* if there is at least one column of A with nonzero entries in both rows. If each row of A is considered as a vertex in an undirected graph with two vertices connected by an edge whenever the corresponding rows conflict, then the (maximum) GUB problem becomes one of finding a (maximum) independent set of vertices in the graph. An independent set of vertices

in a graph is a subset of the total vertex set with no two vertices adjacent (connected by an edge) in the graph.

The conflict matrix representation of the GUB problem (e.g. [10]) uses an $m \times m$ symmetric binary matrix M with each row and column representing a row of A. M has +1 values in those i, j entries where row i and row j conflict in A. By definition, every row conflicts with itself so the main diagonal of M has all +1 entries. The (maximum) GUB problem then becomes one of finding (through permutation of the rows of A) an embedded identity matrix (of maximum size) in the conflict matrix M.

The vector space representation [16] considers each row of K as a vector in n-space having unit length in those directions corresponding to its non-zero entries. The vector R is formed as the sum of each of the row vectors. A unit hypercube in n-space situated at the origin with length 1 in all positive directions represents the feasible GUB region. If R extends beyond this region, the set of rows is not a GUB set and at least one row must be removed to bring R into the feasible region. The (maximum) GUB problem becomes one of determining (the minimum number of) rows which must be removed in order to bring R into the feasible region. The heuristics based on this representation compute gradient vectors which indicate the direction of shortest distance to the feasible region and remove first those rows which produce the greatest movement in that direction; these methods produce GUB sets of comparable quality to other heuristics, but have proven to be more computationally efficient.

The (maximum) Embedded Generalized Network problem for (L) can be stated:

(GN) Find a (maximum) subset of rows in A which exhibit the property that each column of A has at most two nonzero entries in the subset.

The (maximum) GN problem can be formulated as a binary integer program. Defining K as before:

$$\text{(GNI)} \quad \text{maximize} \quad z_1 + z_2 + \cdots + z_m$$

$$\text{s.t.} \quad \sum_i k_{ij} z_i \leq 2, \quad j = 1, \ldots, n,$$

$$\text{where} \quad z_i \in \{0, 1\}.$$

(z_i is an indicator variable for inclusion in the GN set.)

The GUB gradient method is trivially modified to seek GN rows by seeking resultant vectors within a hypercube with sides of length two [4].

The (maximum) Embedded Pure Network problem for (L) can be stated as:

(NET) Find a (maximum) subset of rows in A which can be scaled to contain only $0, \pm 1$ entries and which exhibit the property that each column of A (restricted to those rows) has at most two nonzero entries, and if the column has two nonzero entries, the (scaled) entries are of opposite sign.

Considering, for the moment only, matrices with $0, \pm 1$ entries (or a subset of m rows with $0, \pm 1$ entries in a general matrix) with no scaling allowed, the (maximum) NET problem can be formulated as the binary integer program:

$$\text{(NETI)} \quad \text{maximize} \quad z_1 + z_2 + \cdots + z_m$$

$$\text{s.t.} \quad \sum_{i:a_{ij}=-1} z_i \leq 1, \quad j = 1, \ldots, n,$$

$$\sum_{i:a_{ij}=+1} z_i \leq 1, \quad j = 1, \ldots, n,$$

$$\text{where} \quad z_i \in \{0, 1\}.$$

(z_i is an indicator variable for inclusion in the pure network set.)

Unfortunately, NET does not lend itself to the many representations which GUB admits. The primary reason for this is that the scaling problems associated with NET make it impossible to disregard the real values of the nonzero coefficients in A. Also, the concept of pairwise row conflicts so useful in the GUB algorithms does not apply directly to network rows when row scaling is allowed.

To efficiently confront the scaling dilemma, we restrict the eligibility of rows for membership in the network set. The most obvious restriction is to allow no scaling and consider only those rows with intrinsic $0, \pm 1$ entries. Two less restrictive options are employed in the algorithms described later. These are:

1. Admit only rows with intrinsic $0, \pm 1$ entries but allow row *reflection* (multiplication of a row by -1).

2. Admit only rows whose nonzero entries can be row-scaled to $0, \pm 1$. This includes rows with all nonzero entries of the same absolute value.

Two representations of the NET problem are developed for the algorithms presented.

As suggested by Thomen [17], GUB heuristics can be used to produce a bipartite-network row factorization which can be partitioned into two subsets, G_1 and G_2, such that each column of the matrix has at most one non-zero entry in G_1 and at most one non-zero entry in G_2. Additionally, the entries must be of opposite sign. To produce such a (D-GUB) factorization, a GUB heuristic can be applied to the eligible rows of A producing G_1, and then applied again to remaining eligible rows (not selected in the first pass and compatible for NET inclusion, allowing row reflection if necessary) giving G_2.

If we consider only the rows of A with $0, \pm 1$ entries, or those which have been scaled to $0, \pm 1$, a vector space representation for NET can be developed similar to that developed for GUB. The representation can also allow reflection of rows, if desired.

With each row in the eligible set, we associate two vectors in n-space, V_i^+ and V_i^-, each consisting of ± 1 in those dimensions corresponding to ± 1 entries in row i and zero in all other dimensions. For example, if row i is $(1, 0, -1, 1, 0)$, then $V_i^+ = (1, 0, 0, 1, 0)$ and $V_i^- = (0, 0, -1, 0, 0)$.

We define R^+ (R^-) as the resultant vector from the sum of all V_i^+ (V_i^-). These vectors extend from the origin into the orthants of n-space corresponding to all positive dimensions and all negative dimensions, respectively. A unit hypercube in each of these orthants constitutes the feasible NET region. Should either R^+ or R^-

extend beyond its feasible region then the rows in the eligible set do not currently form an admissible set of network rows.

The reflection (multiplication by -1) of a row merely results in the switching of the V_i^+ and V_i^- vectors for the row. That is, when row i is reflected, the negative of V_i^+ becomes V_i^- and the negative of V_i^- becomes V_i^+. This in turn will change the vectors R^+ and R^-. In fact, it is possible that just the reflection of these rows in an infeasible set may bring R^+ and R^- into their feasible regions without deletion of any rows.

If either R^+ or R^- extends beyond the feasible region, a row penalty for each row is computed as the dot product of V_i^+ and R^+ plus the dot product of V_i^- and R^-. The row with the greatest row penalty is identified and the revised penalty for that row, if reflected, is computed. If this reflected penalty is less than the original row penalty, the row is reflected, otherwise it is deleted. When both R^+ and R^- fall within the feasible region, the set of rows which remain constitutes an admissible network set [18].

3. Implementation of two automatic pure network identification heuristics

We describe two algorithms for identifying pure network rows: D-GUB and E-NET. D-GUB is quite fast. E-NET is not as fast but tends to find more pure network rows. D-GUB finds bipartite pure network rows and E-NET seeks general pure networks.

The D-GUB *algorithm*:

Step 0. *Determine eligible rows.* Using the scaling scheme desired, determine which rows of the matrix are eligible for selection as network rows.

Step 1. *Find first* GUB *set.* Apply a GUB heuristic to the eligible set.

Step 2. *Determine eligibility for second* GUB *set.* For each eligible row not included in the first GUB set, check the columns in which the row has nonzero entries. In each of these columns, if the first GUB set has no nonzero entries or one nonzero entry of opposite sign then the row is eligible for inclusion in the second GUB set in its present form. If the first GUB set has no nonzero entries or a nonzero entry of like sign in each column, then the row is eligible for inclusion in reflected form. Otherwise, the row is not eligible and is discarded.

Step 3. *Find Second* GUB *Set.* If there are any rows eligible for the second pass, reapply the GUB heuristic to those rows.

The D-GUB Algorithm used in Steps 1 and 2 is the two-phase, one-pass, nonbacktracking GUB algorithm described in [3]. Phase I attempts to delete as few rows as possible in order to produce a feasible GUB set. Phase II examines the rows deleted in Phase I and reincludes rows which do not violate the GUB restriction.

Computational experience with many real-world models indicates that Phase II of the GUB heuristic rarely adds additional rows to the GUB sets obtained in either pass. For the second GUB set, Phase II is especially ineffectual.

The E-NET Algorithm begins with an eligible set of rows which normally do not form an admissible network set and attempts to delete as few rows as possible to obtain a feasible set. Deleted rows are then considered for reinclusion if they do not violate the feasibility requirements.

The measure of infeasibility at any point is a matrix penalty computed as the sum of individual row penalties. Rows in the eligible set are examined in order of decreasing row penalty and either reflected, if the row penalty would be reduced, or removed and placed in a candidate set for later use. This guarantees that the matrix penalty will be reduced by at least 1 at each iteration. Thus, the number of iterations in Phase I is limited by the initial matrix penalty, which is *polynomially* bounded. In Phase II, the rows in the candidate set are examined for reinclusion in the eligible set if they do not increase the matrix penalty. Those not reincluded are discarded.

Statement of the problem:

Let $A=\{a_{ij}\}$ be an $m \times n$ matrix with $a_{ij}=0, \pm 1\ \forall i, j$.

Problem: Find a matrix $N=\{n_{ij}\}$ with $(m-k)$ rows and n columns which is derived from A by

1. Deleting k rows of A where $k \geq 0$,

2. Multiplying zero or more rows of A by -1, where N has the property that each column of N has at most one $+1$ element and at most one -1 element. We wish to find a large N in the sense of containing as many rows as possible (i.e. minimize k).

Terminology and notation:

1. E is the set of row indices for rows eligible for inclusion in N and is called the eligible set.

2. C is the set of row indices for rows removed from E in Phase I (Deletion). Some rows in C may be readmitted to E in Phase II. C is called the candidate set.

3. The phrase 'reflect row i' of A' means to multiply each element in row i' by -1 (i.e. $a_{i'j} \leftarrow -a_{i'j}\ \forall j$).

4. Other notation will be defined in the algorithm itself.

The E-NET *Algorithm*:

Phase I – *Deletion of infeasible rows*

Step 0. *Initialization.* Set $E=\{1, 2, \ldots, m\}$, $C=\emptyset$. For each column j of A compute the + penalty (K_j^+) and the − penalty (K_j^-) as follows:

$$K_j^+ = \left(\sum_{i \in E: a_{ij} > 0} 1 \right) - 1, \qquad K_j^- = \left(\sum_{i \in E: a_{ij} < 0} 1 \right) - 1.$$

These penalties represent the number of excess $+1$ and -1 elements, respectively, in column j which prevent the rows in E from forming a valid N matrix. A penalty value of -1 for K_j^+ (K_j^-) indicates that the column does not contain a $+1(-1)$ element.

Step 1. *Define row penalties.* For every $i \in E$, compute a row penalty (p_i) as follows:

$$p_i = \sum_{j:a_{ij}>0} K_j^+ + \sum_{j:a_{ij}<0} K_j^-.$$

This is simply the sum of + penalties for all columns in which row i has a +1 plus the sum of − penalties for all columns in which row i has a −1.

Step 2. *Define matrix penalty.* Compute the penalty (h) for the matrix by summing the row penalties as follows:

$$h = \sum_{i \in E} p_i.$$

If $h = 0$, then go to Step 7. Otherwise, go to Step 3.

Step 3. *Row selection.* Find the row $i' \in E$ with the greatest penalty, i.e.

$$\text{Find } i' \in E \quad \text{such that} \quad p_{i'} = \max_{i \in E} p_i.$$

(If there is a tie, choose i' from among the tied values.) Compute the reflected row penalty $\bar{p}_{i'}$ for i' as follows:

$$\bar{p}_{i'} = \sum_{j:a_{i'j}>0} (K_j^- + 1) + \sum_{j:a_{i'j}<0} (K_j^+ + 1).$$

This would be the row penalty for row i' if it were to be reflected.

Step 4. *Delete, or reflect row.*

Case (i) $\bar{p}_{i'} \geq p_{i'}$. Let $E \leftarrow E - \{i'\}$, $C \leftarrow C \cup \{i'\}$. Go to Step 5.

Case (ii) $\bar{p}_{i'} < p_{i'}$. Reflect row i'. Go to Step 6.

Step 5. *Reduce column penalties* as follows:

For all j such that $a_{i'j} > 0$, $K_j^+ \leftarrow K_j^+ - 1$.

For all j such that $a_{i'j} < 0$, $K_j^- \leftarrow K_j^- - 1$.

Go to Step 1.

Step 6. *Change column penalties* as follows:

Using the $a_{i'j}$ values *after* reflection of row i':

For all j such that $a_{i'j} > 0$, $K_j^+ \leftarrow K_j^+ + 1$ and $K_j^- \leftarrow K_j^- - 1$.

For all j such that $a_{i'j} < 0$, $K_j^+ \leftarrow K_j^+ - 1$ and $K_j^- \leftarrow K_j^- + 1$.

Go to Step 1.

Phase II – *Reinclusion of rows from C*

Step 7. *Eliminate conflicting rows.* The rows in E, some possibly reflected from the orignal A matrix, form a valid N matrix. However, some of the rows removed from E and placed in C may now be reincluded in E if they do not make $h > 0$.

Remove from C (and discard) all rows which, if reincluded in E in present or reflected form, would make $h > 0$, i.e. remove i from C if

(a) $\exists j_1$ such that $a_{ij_1} > 0$ and $K_{j_1}^+ = 0$

or $a_{ij_1} < 0$ and $K_{j_1}^- = 0$

and

(b) $\exists j_2$ such that $a_{ij_2} > 0$ and $K_{j_2}^- = 0$

or $a_{ij_2} < 0$ and $K_{j_2}^+ = 0$

If $C = \phi$, *STOP*, otherwise go to Step 8.

Step 8. *Select row for reinclusion.* At this point a row from C may be reincluded in E. There are several possible schemes for selecting the row. After the row is reincluded, the column penalties are adjusted. Then go to Step 7.

No dominating rule has been discovered for breaking ties in maximum row penalty encountered in Step 3. The rule used herein is to select the row with the minimum number of nonzero entries in an attempt to place a larger number of nonzero entries in the network set. Other possible rules are 'first come, first served', maximum number of nonzero entries, type of constraint, or modeler preference.

Although the algorithm described above is presented for a matrix with strictly $0, \pm 1$ entries, it can be generalized to any matrix by simply letting E be the set of rows with strictly $0, \pm 1$ entries or which can be scaled to contain only $0, \pm 1$ entries.

Prespecified network rows can also be accommodated with the following modifications:

Let $P = \{i | \text{row } i \text{ is prespecified}\}$.

Then $E \leftarrow E - P$.

After computation of K_j^+ and K_j^- in Step 0, for each column j,

if $\exists\ i \in P$ such that $a_{ij} = 1$ then $K_j^+ \leftarrow K_j^+ + 1$,

if $\exists\ i \in P$ such that $a_{ij} = -1$ then $K_j^- \leftarrow K_j^- + 1$.

Rows in P are not eligible for deletion or reflection. At the termination of the algorithm, the rows in N are given by $E \cup P$.

Computational experience on real-world models indicates that Phase II of the E-NET algorithm is even less productive than that of the GUB algorithm. In only two of sixteen cases were any rows eligible for reinclusion and the maximum number eligible was three. This indicates that the expense of examining the rows in the candidate set for eligibility is probably not justified for the occasional small improvement in quality.

4. Problem complexity

Analysis of the NET problem suggests that it cannot be solved optimally by an efficient algorithm.

The problem of entire conversion by general linear transformation of any matrix to the node-arc incidence matrix of a pure network *if* such conversion is possible, has been shown to be polynomial in complexity [1, 14]. This, however, does not apply to the problem of finding the maximum embedded pure network should entire conversion fail. (The *entire* GUB problem is polynomial, too.) A decision problem implied by finding the maximum size embedded pure network set is the following:

(NETD) Given an $m \times n$ matrix A and an integer $p < m$, determine whether A contains a set of p or more rows such that each column of A (restricted to those rows) has at most two nonzero entries equal to ± 1, where two entries in the same column must be of opposite sign.

Given a set of p rows from A, it is easy to verify, in polynomial time, whether the set satisfies the above criterion. Given an integer $p < m$, it is not easy to determine whether there exists a set of p or more rows in A which satisfies the criterion—in general, there does not currently exist an algorithm which can do so in polynomial time.

Two rows conflict if they both contain a nonzero element of like sign in a common column. It is evident that the absence of pairwise conflicts is necessary in a valid pure network set, for the existence of a conflict violates the opposite-sign requirement for columns containing two nonzero elements. It is also sufficient, because the violation of the criterion for a valid network set would require at least one column of A to contain at least two nonzero entries of like sign in rows of the set. This, in turn, would imply that the two rows in which this occurs are in conflict.

NETD is obviously in NP. We show that NETD is NP-complete by a transformation from the independent set decision problem which is known to be NP-complete [7]:

(ISD) Given any graph G with m vertices and an integer $p < m$, determine whether G contains a set of p or more independent vertices (i.e. vertices not adjacent).

ISD can be converted (in polynomial time) to an instance of NETD as follows. Create an $m \times n$ vertex-edge incidence matrix A with $a_{ij} = 1$ if edge j is incident to vertex i in G and zero otherwise. There exists a set of p or more nonconflicting rows in A if and only if there exists a set of p or more independent vertices in G. Therefore, NETD in NP-complete.

NETD with row-reflection allowed is also NP-complete. Create A from G as above and append an $n \times n$ negative identity matrix below A. Then, with row reflection allowed, $\binom{A}{-I}$ contains a set of $n+p$ *or more* nonconflicting rows if and only if G contains an independent set of size p or greater. Therefore, NETD with row-reflection is NP-complete.

This analysis of pure network identification algorithms has only addressed the worst-case bound. No conclusions can be made about the average performance of an optimal algorithm—it may be possible to develop an optimal algorithm with good average performance, but having an exponential worst-case bound.

5. Upper bounds on maximum pure network set size

The problem of finding a maximum-size pure-network set of rows in a matrix appears to be hard. This also applies to the problem of determining just the size of

a maximum set. Upper bounds on the maximum set size, computed in polynomial time, can be useful in evaluating the quality of network sets produced by heuristic algorithms.

The bounds developed here apply to the maximum set size obtainable from the set of eligible rows, and thus depend on the scaling restrictions employed.

Each column of the matrix is allowed at most two nonzero entries in the network set. If k represents the maximum number of nonzero entries in any column of A (considering only entries in eligible rows), then it is clear that at least $k-2$ rows must be deleted from the eligible set in order to make this 'worst column' feasible. Since the column counts are readily available in the form of the column penalties (K_j^+ and K_j^-), an upper bound on the network set size for a matrix with m eligible rows is:

$$u_1 = m - \max_j (K_j^+ + K_j^-).$$

This bound is *sharp* in that matrices can be constructed for which it is achieved.

A second bound almost always tighter than u_1, is based on a matrix penalty computed from column penalties, rather than row penalties as in the NET algorithm. This penalty is:

$$H = \sum_{j:K_j^+>0} K_j^+ + \sum_{j:K_j^-<0} K_j^-.$$

Clearly, as long as $H>0$, the rows remaining in the eligible set do not form a valid network set. The reflection of a row in the eligible set may decrease H, increase H, or leave it unchanged. The deletion of a row from the eligible set may decrease H, or leave it unchanged. The actual effect of a reflection or deletion depends on the rows remaining in the eligible set and their state (unreflected or reflected) at the time. However, it is possible to compute for each row the maximum possible reduction in H obtainable by reflection or deletion of the row, regardless of the other rows remaining in the eligible set. These maximum possible reductions are called the *reflection potential* and *deletion potential* for the row, respectively.

The bound is determined by finding the minimum number of row deletions necessary to reduce H to zero. This cannot, of course, be specified exactly; however, the result will be conservative in that it will guarantee that at least that number of rows must be deleted.

Consider the possible states of a column j of A in which row i has a nonzero entry (i.e. $a_{ij} \neq 0$). The six possible cases are summarized in Table 1.

The nonzero entries in each column are counted only when they occur in the initial eligible set. The penalties used are those computed before any row reflections or deletions have occurred.

Consider first the effect on column j, and thus H, of reflecting row i. In cases 1, 5 and 6, reflection of row i would not change H. In case 4, reflection of row i would decrease H by 1, unless another row with a nonzero in column j was previously reflected. In cases 2 and 3, reflection of row i would actually increase H by 1, unless

Table 1
K_j^l = column penalty of like sign to a_{ij} (K_j^+ if $a_{ij}>0$; K_j^- if $a_{ij}<0$); K_j^u = column penalty of unlike sign to a_{ij}

Case	K_j^l	K_j^u
1	0	−1
2	0	0
3	0	>0
4	>0	−1
5	>0	0
6	>0	>0

enough other rows with nonzero entries in column j were reflected or deleted to produce a −1 value for K_j^u. Since we cannot be sure that reflection in cases 2 and 3 would actually increase H, we must consider H unchanged by reflection in these cases. In summary, we allow H to be decreased only by reflection of rows with nonzero entries in columns exhibiting case 4. The reflection potential for row i is computed by summing the effects for each column in which row i has a nonzero element, with the condition that only one row reflection is allowed to decrease H for each column exhibiting case 4.

Row deletions provide greater opportunity for reducing H. In cases 1 and 2, deletion of row i has no effect on H, while in cases 4, 5, and 6, deletion of row i directly decreases H by 1. In case 3, deletion of row i does not directly decrease H, but it allows reflection of another row with a nonzero in column j, producing a net decrease of 1 in the value of H. In summary, we allow H to be decreased by deletion of rows with nonzero entries in columns exhibiting case 3, 4, 5 or 6. The deletion potential for row i is computed by summing the effects for each column in which row i has a nonzero entry.

To obtain this bound, the reflection and deletion potentials for each row in the eligible set are computed. Then the maximum possible reduction of H by row reflections alone is computed by summing the individual row reflection potentials. If $H>0$ at this point, then rows must be deleted. Rows are deleted in order of decreasing deletion potential until $H \leq 0$. The upper bound is then computed as:

$$u_2 = m - \text{number of rows deleted},$$

where m is the number of rows in the initial eligible set.

This bound is evidently sharp, because examples can be constructed which satisfy the bound exactly.

The u_2 bound is tighter than the u_1 bound for the real-world models we have examined, however counter examples exist.

Similar arguments can be used to construct even better bounds, but the additional computation cost may not be justified for routine use with every model.

6. Computational results

The D-GUB and E-NET algorithms were coded in FORTRAN and were tested on the set of real-world models with characteristics shown in Table 2. The counts of nonzero coefficients do not include objective functions or right-hand sides.

Table 2
Sample LP (MIP) model characteristics

Model	Description	Rows	Columns		Nonzero coefficients
			Total	Binary	
NETTING	Currency exchange	90	177	114	375
AIRLP	Distribution	171	3040	0	6 023
COAL	Energy development	171	3753	0	7 506
TRUCK	Fleet dispatch (Set covering)	220	4752	4752	30 074
CUPS	Production scheduling	361	582	145	1 341
FERT	Production & distribution	606	9024	0	40 484
PIES	Energy production & consumption	663	2923	0	13 288
PAD	Energy production & consumption	695	3934	0	13 459
ELEC	Energy production & consumption	785	2800	0	8 462
GAS	Production scheduling	799	5536	0	27 474
PILOT	Energy production & consumption	976	2172	0	13 057
FOAM	Production scheduling	1000	4020	42	13 083
LANG	Equipment & manpower scheduling	1236	1425	0	22 028
JCAP	Production scheduling	2487	3849	560	9 510
PAPER	Econometric production	3529	6543	0	32 644
ODSAS	Manpower planning	4648	4683	0	30 520

The results obtained for the D-GUB algorithm are given in Table 3. The row eligibility criterion used was that each contain only $0, \pm 1$ entries, or be able to be scaled to $0, \pm 1$ entries by row scaling only. The number of eligible rows as a fraction of the total row count ranged from 9% to 100% (the objective row(s) not being eligible in any case). The number of GUB rows obtained in each pass is indicated. In two cases, the entire eligible set was determined to be a GUB set, so no second pass was required. The times given are in CPU seconds for the IBM 360/67 with the program compiled using FORTRAN IV H (Extended) with OPTIMIZE (2).

The results for the E-NET algorithm are given in Table 4. Also included are the upper bounds on the maximum pure network set size computed from the problem data. The times given for determining the eligible set should be nearly the same as those for the D-GUB algorithm since the same eligibility criterion and code were used in both cases. The eligibility of rows in the candidate set for reinclusion in Phase II was determined, but Phase II was not included due to the absence of eligible rows in nearly every case. The solution time does not include the time required to determine eligibility for Phase II. The NET quality value is the number

Table 3
D-GUB algorithm results

Model	Eligibility		Network rows found						Rows Reflected	Nonzero coefficients in set
	Rows	Time	Pass 1	Time	Pass 2	Time	Total	Time		
NETTING	59	0.01	36	0.03	18	0.04	54	0.07	18	89
AIRLP	150	0.09	150	0.41	All GUB		150	0.41	0	3000
COAL	111	0.11	111	0.50	All GUB		111	0.50	0	3753
TRUCK	219	0.76	29	3.96	18	4.44	47	8.40	18	1755
CUPS	300	0.05	150	0.14	101	0.15	251	0.29	1	710
FERT	585	0.62	559	3.00	13	3.03	572	6.03	13	16291
PIES	142	0.09	128	0.51	0	0.05	128	0.56	0	1392
PAD	174	0.10	160	0.52	0	0.06	160	0.58	0	1552
ELEC	322	0.12	266	0.47	6	0.52	272	0.99	6	2691
GAS	752	0.58	607	2.61	75	2.39	682	5.00	75	9008
PILOT	109	0.10	96	0.44	13	0.48	109	0.92	1	479
FOAM	966	0.33	917	0.95	34	0.94	951	1.89	1	8001
LANG	850	0.26	342	2.91	243	0.83	585	3.74	1	1804
JCAP	1811	0.26	517	1.49	357	1.01	874	2.50	201	2622
PAPER	2324	0.79	1016	3.53	468	3.71	1484	7.24	433	8176
ODSAS	410	0.61	195	1.84	122	1.55	317	3.39	92	5344

Table 4
E-NET algorithm results

Model	Eligibility		Upper bounds		Network rows found			Rows reflected	Nonzero coefficients in set
	Rows	Time	U1	U2	Number	Time	Quality		
NETTING	59	0.01	58	57	54	0.08	94.74%	18	89
AIRLP	150	0.09	150	150	150	0.35	100%	0	3000
COAL	111	0.11	111	111	111	0.43	100%	0	3753
TRUCK	219	0.76	214	137	46	19.82	33.58%	18	1781
CUPS	300	0.05	299	297	295	0.14	99.33%	1	862
FERT	585	0.62	584	572	572	6.15	100%	13	16291
PIES	140	0.09	139	132	128	0.59	96.97%	0	1392
PAD	174	0.10	171	164	160	0.59	97.56%	0	1552
ELEC	322	0.14	310	306	286	2.07	93.46%	34	2915
GAS	752	0.60	750	710	668	9.71	94.08%	33	11002
PILOT	109	0.10	109	109	109	0.36	100%	1	479
FOAM	966	0.34	965	955	951	1.16	99.58%	1	8001
LANG	850	0.29	836	758	661	14.82	87.20%	2	2239
JCAP	1811	0.26	1801	1092	917	44.07	83.97%	200	2540
PAPER	2324	0.66	2316	2072	1627	94.16	78.52%	603	8995
ODSAS	410	0.61	406	369	286	14.55	77.51%	45	6207

of rows in the network set, expressed as a percentage of the better upper bound on the pure network set size. As explained earlier, the actual maximum network set size is, in general, unknown and thus the actual NET quality may be better than this conservative estimate. In particular, the bounds are almost certainly too high for problems with a large number of eligible rows (e.g. PAPER) and for problems with dense, or unstructured coefficient matrices (e.g. TRUCK, which is included in this study as a deliberate torture-test).

7. Conclusion

The identification algorithms are very fast (especially when compared with computer time expended in any attempt to solve these problems) and they consistently produce maximum or near maximum pure network sets (from the eligible rows) as evidenced by the upper bounds.

Better yet, they provide independent insights which can be used to explain and improve the model at hand, or make it easier to solve. For instance, several models in Table 2 have been revealed as multi-commodity production/transportation problems, a totally unexpected perspective for the model proponents. Further, these results have yielded prescriptive benefits for model solution, especially via decomposition.

Many problems exhibiting intrinsic network structure are disguised by their formulation and resist the simplistic attempts used here to rescale them. In particular, the COAL model is known to be an entire network if appropriately restated, but it is not yet evident how this is to be discovered using efficient, general, problem-independent automatic identification. Methods used to scale an entire matrix to $0, \pm 1$ values (see [1, 14]) can be attempted, but failing entire conversion the next step is not evident.

Using the conflict matrix method of Greenberg and Rarick [10], Schrage [15] reports finding several other embedded structures (e.g. pre-Leontief rows). Our implementation and experience with this method at large scale has not been encouraging. Its principal disadvantage is the requirement for some efficient representation of the conflict matrix. We feel that the superior speed and modest region demands of the gradient method exhibited in GUB, GN and NET identification will carry over to the identification of other special structures. In particular, we have successfully sought pre-Leontief and network decomposition factorizations, the former were sought via an obvious redefinition of a conflict in the GUB gradient method, and the latter with repeated use of NET identification and column dropping.

McBride [12] has employed these identification methods in his 'side constrained network' simplex implementations. He reports that computation time and cost for GN, or NET identification are more than offset by reduced solution effort required by his factorized simplex system.

8. Acknowledgment

We enjoyed scrupulous reviews of this paper by the referees and R. Kevin Wood. A referee's suggestion (attributed to Gary Koehler) and a spirited session with Kevin eventually accommodated row reflection in the complexity argument.

References

[1] R. Bixby and W. Cunningham, "Converting linear programs to network problems", *Mathematics of Operations Research* 5 (1980) 321–357.
[2] A. Brearly, G. Mitra and H. Williams, "Analysis of mathematical programming models prior to applying the simplex algorithm", *Mathematical Programming* 8 (1975) 54–83.
[3] G. Brown and D. Thomen, "Automatic identification of generalized upper bounds in large-scale optimization models", *Management Science* 26 (1980) 1166–1184.
[4] G. Brown and R. McBride, "Extracting embedded generalized network problems from general LP problems", paper presented at ORSA/TIMS (Houston, TX, September 1981).
[5] G. Brown and W. Wright, "Automatic identification of embedded structure in large-scale optimization models", in: H. Greenberg and J. Maybee, eds., *Computer-assisted analysis and model simplification* (Academic Press, New York, 1981) (proceedings from Boulder, CO, March 28, 1980) pp. 369–388.
[6] G. Dantzig and R. Van Slyke, "Generalized upper bounding techniques", *Journal of Computer and System Sciences* 1 (1967) 213–226.
[7] M. Garey and D. Johnson, *Computers and intractability: A guide to the theory of* NP-*completeness* (W.H. Freeman and Company, San Francisco, 1979).
[8] G. Graves and R. McBride, "The factorization approach to large-scale linear programming", *Mathematical Programming* 10 (1976) 91–110.
[9] G. Graves and T. Van Roy, "Decomposition for large-scale linear and mixed integer linear programming", UCLA Technical Report (Los Angeles, CA, November 1979).
[10] H. Greenberg and D. Rarick, "Determining GUB sets via an invert agenda algorithm", *Mathematical Programming* 7 (1974) 240–244.
[11] R. McBride, "Linear programming with linked lists and automatic guberization", Working Paper No. 8175, University of Southern California, School of Business (Los Angeles, CA, July 1975).
[12] R. McBride, "Solving linear programs with network factorization", Working Paper, University of Southern California, School of Business (Los Angeles, CA, September 1982).
[13] V. Klee, "Combinatorial optimization: What is the state of the art", *Mathematics of Operations Research* 5 (1980) 1–26.
[14] J. Musalem, "Converting linear models to network models", Ph.D. Dissertation, UCLA (Los Angeles, CA, January 1980).
[15] L. Schrage, "Some comments on hidden structure in linear programs", in: H. Greenberg and J. Maybee, eds., *Computer-assisted analysis and model simplification* (Academic Press, New York, 1981) (proceedings from Boulder, CO, March 28, 1980) pp. 389–395.
[16] S. Senju and Y. Toyoda, "An approach to linear programming with 0–1 variables", *Management Science* 15 (1968) B196–B207.
[17] D. Thomen, "Automatic factorization of generalized upper bounds in large-scale optimization problems", M.S. Thesis, Naval Postgraduate School (Monterey, CA, September 1979).
[18] W. Wright, "Automatic identification of network rows in large-scale optimization models," M.S. Thesis, Naval Postgraduate School (Monterey, CA, September 1980).