Mathematical Programming 32 (1985) 11-31
North-Holland

EXTRACTING EMBEDDED GENERALIZED NETWORKS FROM LINEAR PROGRAMMING PROBLEMS

Gerald G. BROWN
Naval Postgraduate School, Monterey, CA 93943 USA
Richard D. McBRIDE
University of Southern California, Los Angeles, CA 90089-1421 USA
R. Kevin WOOD
Naval Postgraduate School, Monterey, CA 93943 USA

Received 10 August 1983
Revised manuscript received 8 November 1984

If a linear program (LP) possesses a large generalized network (GN) submatrix, this structure can be exploited to decrease solution time. The problems of finding maximum sets of GN constraints and finding maximum embedded GN submatrices are shown to be NP-complete, indicating that reliable, efficient solution of these problems is difficult. Therefore, efficient heuristic algorithms are developed for identifying such structure and are tested on a selection of twenty-three real-world problems. The best of four algorithms for identifying GN constraint sets finds a set which is maximum in twelve cases and averages 99.1% of maximum. On average, the GN constraints identified comprise more than 62.3% of the total constraints in these problems. The algorithm for identifying embedded GN submatrices finds submatrices whose sizes, rows plus columns, average 96.8% of an LP upper bound. Over 91.3% of the total constraint matrix was identified as a GN submatrix in these problems, on average.

"The act of being wise is the act of knowing what to overlook."
William James (ca. 1890)

Key words: Linear Programming, Generalized Networks, Basis Factorization, Computational Complexity, Heuristic Algorithms

1. Introduction

Large-scale linear programming (LP) models frequently have sparse coefficient matrices with special structure. If special structure can be identified, it can often be exploited to reduce the cost of solving the LP. 'Direct factorization', e.g. [13], maintains a partitioning of the rows and/or columns of all simplex bases. Computations are reduced with respect to standard methods if special structure can be isolated within the partitions. 'Decomposition', e.g. [14], splits a problem into a master problem and one or more subproblems. This technique is most efficient when subproblems consist entirely of special structure allowing their rapid solution. The details of these exploitation schemes will not be discussed here.

Useful structures found embedded in a subset of the rows and/or columns of an LP constraint matrix include *simple upper bounds* (at most one nonzero element in each row), *generalized upper bounds* (GUB) (at most one nonzero coefficient in each column), and *networks* (at most two nonzero elements in each column). Varieties of embedded networks include the general case, *generalized networks* (GN), *generalized transshipment networks* (GT) (at most one coefficient not equal to +1), and *pure networks* (NET) (at most one +1 and one −1 in each column).

Simple upper bounds, GUB and NET structures have been exploited in various commercial and experimental optimization systems, and efficient automatic identification schemes have been developed to find these structures, e.g., [4, 7, 8].

Recent research has produced very efficient specialized simplex algorithms for solving network problems. (For example, see [3] for NET, [6] for GN, and [6, 11] for GT.) This research has, in turn, been exploited to develop factorized optimization systems which solve general LP problems with a set of rows exhibiting NET structure [12], GN structure [18], and GT structure [12, 19]. Even more recently, optimization systems have been tested which use direct factorization [19] or primal and/or dual decomposition [14] to exploit embedded GN structure.

Now that software is available to solve GN (and GT) problems [6], it is very likely that several research groups will exploit GN in various ways in the near future. To support this research, we are interested in efficiently and automatically identifying GN structure of the following varieties in general LP coefficient matrices:

GN_C A subset of LP columns which are GN, or

GN_R A subset of LP rows which are GN, or

$GN_{R,C}$ An embedded GN within a subset of the rows and columns of LP.

Because the efficiency of solving a general LP with GN-exploiting methods is enhanced if the GN structure is large, *maximum* GN structures are our goal. This leads to the maximization problems described below.

Let $A = \{a_{ij}\}$ be the $m \times n$ coefficient matrix of LP, and let $H = \{h_{ij}\}$ be the associated 0–1 incidence matrix for A. The three maximization problems, formulated as integer programs, are

$M(GN_C)$:

$$\max_{C} \sum_j c_j$$

$$\text{s.t.} \quad \sum_i h_{ij} c_j \leq 2 \quad \text{for all } j,$$

$$c_j \in \{0, 1\},$$

where c_j is a binary decision variable indicating inclusion of column j in GN_C;

$M(GN_R)$:

$$\max_{R} \sum_i r_i$$

$$\text{s.t.} \quad \sum_i h_{ij} r_i \leq 2 \quad \text{for all } j,$$

$$r_i \in \{0, 1\},$$

where r_i is a binary decision variable indicating inclusion of row i in GN_R; and

$M(GN_{R,C})$:

$$\max_{R,C} \quad \sum_i r_i + \sum_j c_j$$
$$\text{s.t.} \quad \sum_i h_{ij} r_i + m_j c_j \leq 2 + m_j \quad \text{for all } j,$$
$$r_i, c_j \in \{0, 1\},$$

where r_i and c_j are binary decision variables indicating respective inclusion of row i and column j in $GN_{R,C}$, and where $m_j \equiv \sum_i h_{ij} - 2$. Note that our definitions of maximum GN factorizations are expressed simply as the sum of the rows and/or columns included.

Much work has been done on the development of algorithms to identify special substructures in LPs. Previous work in identifying GUB subsets of constraints is well known [4, 7]. Brown and Wright [8] have explored ways to identify NET subsets. Extraction of hidden NET structure with general linear transformations has been discussed by Bixby and Cunningham [2] and by Musalem [20]. Identification of GN row sets and other structures has been proposed by Schrage [21].

The problems of identifying maximum GUB and NET constraint subsets are NP-complete and consequently, exact solutions cannot be guaranteed to be obtained quickly. Since GUB and NET constraints are special cases of GN constraints, it is to be expected that exact solutions of the GN identification problems will also be difficult to obtain. We show that the GN identication problems are, in fact, NP-complete, but also give effective and reliable heuristic algorithms for them.

In Section 2, the complexities of the three maximization problems are investigated. $M(GN_R)$ and $M(GN_{R,C})$ are shown to be difficult and so, in Section 3, efficient algorithms are developed for finding approximate solutions to these problems. Four specialized integer programming heuristics are described for identifying maximal GN_R sets. Two of the algorithms are 'addition' heuristics which begin with the empty GN_R set and successively add rows while maintaining feasibility. The other two algorithms are 'deletion' heuristics which begin with an infeasible GN_R set and successively delete rows until a feasible set is found. Algorithm GNRC for $M(GN_{R,C})$ takes as input the GN_R set found by any one of the GN_R heuristics. Then, it successively adds rows which introduce the least amount of weighted infeasibility and drops those columns where an infeasibility results. In this way, a sequence of $GN_{R,C}$ sets is produced and the maximum of these taken to be the heuristic solution to $M(GN_{R,C})$. After the algorithms are presented, computational experience is given in Section 4.

2. Complexity

In this section we investigate the complexity of $M(GN_C)$, $M(GN_R)$, and $M(GN_{R,C})$. $M(GN_C)$ is trivially solvable in polynomial time by choosing all columns with at most two nonzero elements in them; consequently, its complexity will not be discussed further. The other two problems are more interesting.

Following standard practice, $M(\mathrm{GN_R})$ and $M(\mathrm{GN_{R,C}})$ will be studied with respect to their associated decision problems:

$D(\mathrm{GN_R})$: Does there exist a set of rows R in $\boldsymbol{H}$ such that, for positive integer $k<m$,

$$|R| \geq k \quad \text{and} \quad \sum_{i \in R} h_{ij} \leq 2 \quad \text{for all } j?$$

$D(\mathrm{GN_{R,C}})$: Does there exist a set of rows R and columns C in $\boldsymbol{H}$ such that, for positive interger $k<m+n$,

$$|R|+|C| \geq k \quad \text{and} \quad \sum_{i \in R} h_{ij} \leq 2 \quad \text{for all } j \in C?$$

Of course, a polynomial algorithm for one of the above decision problems would imply a polynomial algorithm for the associated maximization problem using, say, a binary search on the values of k.

We consider the complexity of $D(\mathrm{GN_{R,C}})$ first. Yannakakis [24] investigated the problem of finding the least number of nodes which can be deleted from a bipartite graph such that the resulting induced subgraph has a particular property. Restated in terms of the decision problem, he gives the following theorem on 0–1 matrices as a corollary of his results on graphs.

Theorem 1. *Let $\boldsymbol{Q}$ be any class of 0–1 matrices which is closed under permutation and deletion of rows and columns. Let $\boldsymbol{H}$ be an $m \times n$ 0–1 matrix, and let k be some positive integer, $k<m+n$. Then, finding an $m_0 \times n_0$ submatrix $\boldsymbol{H}_0$ of $\boldsymbol{H}$ such that $\boldsymbol{H}_0 \in \boldsymbol{Q}$ and $m_0+n_0 \geq k$ is polynomial if the matrices of $\boldsymbol{Q}$ have bounded rank and is NP-complete otherwise.*

It is assumed above that membership in $\boldsymbol{Q}$ can be determined in polynomial time for a matrix of bounded size (otherwise, NP-hardness would be implied).

This theorem is impressive in that it handles the NP-completeness question for 0–1 matrices in a wholesale fashion. The NP-completeness of $D(\mathrm{GN_{R,C}})$ follows as a simple corollary.

Corollary 1. *$D(\mathrm{GN_{R,C}})$ is NP-complete.*

Proof. Let $\boldsymbol{Q}$ be the class of 0–1 matrices with at most two 1s in each column. $\boldsymbol{Q}$ is obviously closed under permutation and deletion of rows and columns: matrices of arbitrarily large rank can be found in $\boldsymbol{Q}$ and membership in $\boldsymbol{Q}$ can be determined in polynomial time. $D(\mathrm{GN_{R,C}})$ for the incidence matrix $\boldsymbol{H}$ is equivalent to searching for an $m_0 \times n_0$ submatrix $\boldsymbol{H}_0$ of $\boldsymbol{H}$ such that $\boldsymbol{H}_0 \in \boldsymbol{Q}$ and $m_0+n_0 \geq k$. Therefore, by Theorem 1, $D(\mathrm{GN_{R,C}})$ is NP-complete. □

A 0–1 matrix $\boldsymbol{H}$ is represented as a bipartite graph with nodes on one side of the bipartition corresponding to rows, nodes on the other side of the bipartition corresponding to columns, and an edge (i, j) for each $h_{ij} = 1$. $D(\mathrm{GN_R})$ corresponds to a node-deletion problem with deletions restricted to one side of the bipartition; Yannakakis's results do not directly apply since they pertain to node deletions on either side of the bipartition.[1] Therefore, we use a problem-specific proof to show that $D(\mathrm{GN_R})$ is NP-complete.

Lemma 1. *$D(\mathrm{GN_R})$ is NP-complete.*

Proof. For ease of representation, $D(\mathrm{GN_R})$ will be equivalently stated in matrix notation:

$D(\mathrm{GN_R})$: Does there exist a binary m-vector $\boldsymbol{x}$ such that $\mathbf{1}\boldsymbol{x} \geqslant k$ and $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{x} \leqslant \mathbf{2}$?

$D(\mathrm{GN_R})$ is obviously in NP. We show that it is NP-complete by a transformation from the 'Exact Cover by 3-Sets' problem [15], as specialized by Garey and Johnson [10].

$D(\mathrm{X3C})$: Does there exist a binary p-vector $\boldsymbol{y}$ such that $\mathbf{1}\boldsymbol{y} = q$ and $\boldsymbol{N}\boldsymbol{y} = \mathbf{1}$ where $\boldsymbol{N}$ is a $3q \times p$, 0–1 matrix with exactly three 1s in each column and at most three 1s in each row?

For each row i in $\boldsymbol{N}$ with only one 1 or two 1s, augment $\boldsymbol{N}$ with one or two unit vector columns e_i, respectively. Since none of these columns could be included in an exact cover of size q, $D(\mathrm{X3C})$ is equivalent to

$D(\mathrm{X3C'})$: Does there exist a binary vector $\boldsymbol{y}'$ of length $p + l$ such that $\mathbf{1}\boldsymbol{y}' = q$ and $(\boldsymbol{E}, \boldsymbol{N})\boldsymbol{y}' = \mathbf{1}$ where $\boldsymbol{E}$ corresponds to l augmenting columns?

By construction of $D(\mathrm{X3C'})$, no set of columns of cardinality less than q could ever cover all the rows exactly once let alone more than once. Thus, $D(\mathrm{X3C'})$ is equivalent to a 'minimum cover problem'.

$D(\mathrm{MC})$: Does there exist a binary vector $\boldsymbol{y}'$ such that $\mathbf{1}\boldsymbol{y}' \leqslant q$ and $(\boldsymbol{E}, \boldsymbol{N})\boldsymbol{y}' \geqslant \mathbf{1}$?

Let $\boldsymbol{x} = \mathbf{1} - \boldsymbol{y}'$. Since each row contains exactly three 1s, $D(\mathrm{MC})$ is equivalent to a 'maximum uncover problem'.

$D(\mathrm{MUC})$: Does there exist a binary vector $\boldsymbol{x}$ such that $\mathbf{1}\boldsymbol{x} \geqslant p + l - q$ and $(\boldsymbol{E}, \boldsymbol{N})\boldsymbol{x} \leqslant 2$?

Since all above transformations are of polynomial complexity, and since $D(\mathrm{MUC})$ is an instance of $D(\mathrm{GN_R})$, $D(\mathrm{GN_R})$ is NP-complete. □

3. Algorithms

The complexity results of the preceding section indicate that solving $M(\mathrm{GN_R})$ and $M(\mathrm{GN_{R,C}})$ exactly could be very time-consuming. Therefore, heuristic algorithms have been developed for obtaining approximate solutions. We describe the algorithms for $M(\mathrm{GN_R})$ first.

[1] Bartholdi [1] has addressed this topic, but his results are incomplete. For instance, without additional restrictions, his Theorem 2 would imply that $D(\mathrm{GN_C})$ is NP-complete.

$M(GN_R)$ is an integer programming problem of the form $\max cx$ s.t. $Ax \leq b$, x binary, where all data is nonnegative. Thus, integer programming heuristics seem appropriate for attacking this problem. Two basic heuristic techniques exist for solving such integer programs which we label 'addition' heuristics and 'deletion' heuristics. An addition heuristic begins with the feasible solution $x = 0$ and successively sets to 1 that variable x_j which myopically maximizes effective profit. The effective profit associated with x_j is c_j/φ_j, where φ_j is a penalty whose definition varies between heuristics, but which in some way reflects the units of feasibility used up by setting x_j to 1. The addition heuristic stops when no additional variables can be set to 1 without violating feasibility. A deletion heuristic begins with the usually infeasible solution $\mathbf{x} = \mathbf{1}$ and successively sets to 0 that variable x_j which myopically minimizes loss of effective profit c_j/φ_j. Here, φ_j is a penalty which reflects the amount of infeasibility currently being contributed by $x_j = 1$. The deletion heuristic stops when a feasible solution is obtained.

We have specialized two addition heuristics and two deletion heuristics to $M(GN_R)$. The addition heuristics begin with an empty GN_R set and successively add rows to the set until a maximal set is obtained. The deletion heuristics begin with an infeasible GN_R set consisting of all the rows, and rows are successively deleted until a feasible set is obtained. Since a GN_R set obtained by deletion may not be maximal, a second phase, an addition phase, is appended to insure that the set is maximal. To further expand the GN_R set found, it is possible to devise post-maximal techniques similar to the 2-opt, 3-opt and general k-opt procedures used in traveling salesman heuristics, e.g., [16, 17]. Application of such techniques was unwarranted, however, since computational results in Section 4 show that excellent approximate solutions were obtained using the basic addition and deletion heuristics.

The addition heuristics are described by Algorithm GNRa, with variations 'Greedy' and 'Toyoda'. The effective profit associated with adding row i to the GN_R set is $1/RP_i$ where RP_i is a row penalty derived from the current nonmaximal solution, the nonzero elements in the row and feasibility requirements. Thus, at each step of the algorithm, the row with the smallest penalty is added to the GN_R set. Feasibility is maintained by setting to infinity the row penalty of any row whose addition would cause an infeasibility. In the Greedy variation, RP_i equals the number of nonzero elements in the row if the penalty is finite. The Toyoda variation is a modification of an integer programming heuristic developed by Toyoda [23]. In this heuristic, the finite row penalty RP_i is based not only on the number of nonzero elements in the row, but also on how close to feasibility limits addition of the row would bring the current solution.

The deletion heuristics are described by Algorithm GNRd, with variations 'Dobson' and 'Senju & Toyoda'. In this algorithm, each row has a penalty RP_i which, roughly speaking, indicates how much infeasibility the row is contributing. $1/RP_i$ is the loss in effective profit if row i is removed from the GN_R set. Thus, this algorithm successively deletes rows with maximum penalty to minimize the loss of effective profit.

Dobson [9] analyzes and gives worst-case performance guarantees for an addition heuristic for integer programs of the form min cx, s.t. $Ax \geq b, 0 \leq x \leq u$, x *integer*, where all data is nonnegative. By simple substitution of variables, however, the Dobson heuristic may be interpreted as a deletion heuristic for problems in the form of $M(\mathrm{GN_R})$. At each deletion step of this heuristic, RP_i is the number of nonzero elements in row i which are contributing to an infeasibility. If m_0 is the optimal solution to $M(\mathrm{GN_R})$ and m_D is the heuristic solution obtained by deletion only, Dobson's worst-case bound on performance is $(m - m_D)/(m - m_0) \leq \sum_{k=1}^{d} 1/k$ where d is the maximum number of nonzero elements in any row. This is the only performance guarantee known for any of the heuristics implemented in this paper. Unfortunately, the upper bound on m_0 this yields is rather weak in practice. (See Table 3.) Any addition heuristic may be used as a second phase for a deletion heuristic, but for the Dobson deletion heuristic, we chose the greedy addition heuristic as the second phase since the definition of RP_i is consistent between the two phases.

The second variant of GNRd is a specialization of the heuristic devised by Senju and Toyoda [22] which those authors label an 'effective gradient method'. For $M(\mathrm{GN_R})$, H^{T} maps the set of feasible r values into the n-dimensional hypercube whose sides are of length 2. At every step of the algorithm, given current infeasible solution r, $RP_i = (H^{\mathrm{T}}r - 2)^+ h^i$, where the jth element of $(H^{\mathrm{T}}r - 2)^+$ is $\max\{0, \sum_{i=1}^{n} h_{ij} r_i - 2\}$. RP_i may be interpreted as the length of the projection of the vector h^i onto the shortest vector extending from the point $H^{\mathrm{T}}r$ outside of the hypercube to the boundary of the hypercube. The modified Toyoda addition heuristic is used as the second phase of this heuristic.

The two algorithms GNRa and GNRd, with their variations, are outlined as follows:

Algorithm GNRa

Input: The LP coefficient matrix A.

Output: A set of row indices I_{R} corresponding to the largest $\mathrm{GN_R}$ set found in A.

Comment: The basic algorithm is the 'Greedy' addition heuristic. The modified 'Toyoda' heuristic is obtained by substituting the statement in square brackets for its predecessor.

Step 0. 'Initialization'

Initialize:

(a) $I = \emptyset$ and $I' = \{1, 2, \ldots, m\}$.

(b) For each column j, a column bound

$$CB_j = 2.$$

Comment: CB_j is the number of elements column j may contain.

(c) For each $i \in I'$, a row penalty

$$RP_i = \sum_{a_{ij} \neq 0} 1.$$

Step 1. 'Row Addition'

Let $\underline{RP} = RP_s$ be the smallest row penalty (corresponding to row $s \in I'$).

If $\underline{RP} < \infty$ then

(a) Move s from I' to I.

(b) For each column j such that $a_{sj} \neq 0$.

(i) Let $CB_j = CB_j - 1$.

(ii) If $CB_j = 0$ then for each $i \neq s$ such that $a_{ij} \neq 0$, let $RP_i = \infty$.

[(ii) For each $i \neq s$ such that $a_{ij} \neq 0$, if $CB_j = 1$ then let $RP_i = RP_i + 1$, else let $RP_i = \infty$.]

(c) Repeat Step 1.

Step 2. 'Termination'

Print $I_R = I$ and STOP.

End of Algorithm GNRa

Algorithm GNRd

Input: The LP coefficient matrix $\boldsymbol{A}$.

Output: A set of row indices I_R corresponding to the largest GN_R set found in $\boldsymbol{A}$.

Comment: The basic algorithm is the 'Dobson' heuristic. The 'Senju and Toyoda' heuristic is obtained by substituting the statements in square brackets for their predecessors.

Step 0. 'Initialization'

Initialize:

(a) $I = \{1, 2, \ldots, m\}$ and $I' = \emptyset$.

(b) For each column j, a column penalty

$$CP_j = \left(\sum_{\substack{a_{ij} \neq 0 \\ i \in I}} 1 \right) - 2.$$

Comment: CP_j is the number of 'excess' elements in column j.

(c) For each $i \in I$, a row penalty

$$RP_i = \sum_{\substack{a_{ij} \neq 0 \\ CP_j > 0}} 1.$$

Comment: RP_i is number of units of infeasibility which row i is currently contributing.

[(c) For each $i \in I$, a row penalty

$$RP_i = \sum_{\substack{a_{ij} \neq 0 \\ CP_j > 0}} CP_j.$$

Comment: RP_i is the sum of excess elements in columns with a nonzero entry in row i.]

Step 1. 'Row Deletion'

Let $\overline{RP} = RP_l$ be the largest row penalty (corresponding to row $l \in I$).

If $\overline{RP} > 0$ then

(a) Move l from I to I'.

(b) For each column j such that $a_{lj} \neq 0$

(i) If $CP_j = 1$ [If $CP_j > 0$] then, for each $i \neq s$ such that $a_{ij} \neq 0$, let $RP_i = RP_i - 1$.

(ii) Let $CP_j = CP_j - 1$.

(c) Repeat Step 1.

Step 2. 'Row Addition Penalties'

For each $i \in I'$, compute a row penalty

$$RP_i = \begin{cases} \sum_{a_{ij} \neq 0} & \text{if } CP_j < 0 \text{ for all } a_{ij} \neq 0, \\ \infty & \text{otherwise.} \end{cases}$$

$$\left[RP_i = \begin{cases} \sum_{a_{ij} \neq 0} (CP_j + 3) & \text{if } CP_j < 0 \text{ for all } a_{ij} \neq 0 \\ \infty & \text{otherwise.} \end{cases} \right]$$

Step 3. 'Row Addition'

Let $\underline{RP} = RP_s$ be the smallest row penalty (corresponding to row $s \in I'$).

If $\underline{RP} < \infty$, then

(a) Move s from I' to I.

(b) For each j such that $a_{sj} \neq 0$, let $CP_j = CP_j + 1$.

(c) Go to Step 2.

Step 4. 'Termination'

Print $I_R = I$ and STOP.

End of Algorithm GNRd

The execution times of the above algorithms and the other algorithms described in this paper are quite short if proper data structures are used. The initial computation of the row and column penalties can be made very quickly if the nonzero entries in each row and column are stored in a linked list. Column penalties are then updated in a single pass of a row. Because of sparsity, row penalties can usually be updated in passes through just a few columns. Efficiency is further improved if row and column partitions are maintained with an indirect address array which allows contiguous access. Associated with this mapping array, a second array expresses the inverse map to speed updating.

An easily computable upper bound on $M(\mathrm{GN_R})$, denoted UB_R, is useful for checking the efficacy of the above algorithms. Algorithm UBR is designed for this purpose. Let A_1 and A_2 be a partition of the rows of A and let z, z_1 and z_2 be the solutions to $M(\mathrm{GN_R})$ on A, A_1 and A_2, respectively. If UB_1 is any valid upper bound on $M(\mathrm{GN_R})$ for A_1, then

$$z \leq z_1 + z_2 \leq UB_1 + z_2.$$

Algorithm UBR iteratively applies the above statement, computing the simple bound UB_1 and letting $A = A_2$ after each iteration. This is repeated until all columns of A_2 have at most two nonzero elements in them at which point z_2 is equal to the number of rows in A_2. UB_R is then given by the sum of the UB_1 upper bounds found at each iteration plus z_2 found at the last iteration. At each iteration, A is partitioned with respect to that column j having the maximum number of nonzero entries. A_1 is all rows of A with $a_{ij} \neq 0$ and $UB_1 = z_1 = 2$ since column j has only nonzero elements in A_1.

Algorithm UBR

Input: The LP coefficient matrix A.

Output: A value UB_R, an upper bound on $|I_R|$.

Step 0. 'Initialization'

Initialize:

(a) $I = \{1, 2, \ldots, m\}$, and $UB_R = 0$.

(b) For each column j, a column count

$$CC_j = \Big(\sum_{\substack{a_{ij} \neq 0 \\ i \in I}} 1 \Big).$$

Step 1. 'Iterative Partitioning'

Let $\overline{CC} = CC_l$ be the largest column count (corresponding to column l).

If $\overline{CC} > 2$ then

(a) Let $UB_R = UB_R + 2$.

(b) For each $i \in I$ such that $a_{il} \neq 0$,

(i) Delete i from I.

(ii) For each j such that $a_{ij} \neq 0$, update column count letting $CC_j = CC_j - 1$.

(c) Repeat Step 1.

Step 2. 'Termination'

Print $UB_R = UB_R + |I|$ and STOP.

End of Algorithm UBR

Algorithm GNRC, the heuristic for $M(\mathrm{GN}_{R,C})$, is outlined next. Any one of the integer programming heuristics described for $M(\mathrm{GN}_R)$ could be applied to this problem. However, these algorithms will normally give only a single answer to the problem; our algorithm allows the exploration of a complete trajectory of maximal $\mathrm{GN}_{R,C}$ sets beginning with GN_R and ending with GN_C. Our algorithm begins with the set of rows I_R found in Algorithm GNRa or GNRd and repeatedly attempts to expand this set by deleting columns, always saving the largest $\mathrm{GN}_{R,C}$ set found. This approach was suggested by manual analysis of several problems for which the GN_R set is limited by a few key complicating columns. Deleting these columns produced a much larger embedded $\mathrm{GN}_{R,C}$ set, and motivated development of a new factorization LP code which effectively exploits $\mathrm{GN}_{R,C}$ structure [19].

Algorithm GNRC

Input: The LP coefficient matrix A and a GN_R set I_R, $|I_R| < m$, e.g., I_R from Algorithms GNRa or GNRd.

Output: A set of row indices $I_{R,C}$ and a set of column indices $J_{R,C}$ corresponding to the largest $GN_{R,C}$ structure found in A.

Step 0. 'Initialization'

Initialize:

(a) $I = I_R$, $I' = \{1, 2, \ldots, m\} - I$, $J = \{1, 2, \ldots, n\}$, $I_{R,C} = I$, and $J_{R,C} = J$.
Comment: I and J are the current sets of row and column indices while $I_{R,C}$ and $J_{R,C}$ store the best sets found.

(b) For each column $j \in J$, a column penalty

$$CP_j = \Bigg(\sum_{\substack{a_{ij} \neq 0 \\ i \in I}} 1 \Bigg) - 2.$$

Comment: These column penalties remain as an artifact of Algorithm GNRd and can be defined as input.

(c) For each $i \in I'$, a row cost

$$RC_i = \sum_{\substack{a_{ij} \neq 0 \\ CP_j = 0}} 1.$$

Comment: RC_i is the number of columns which must be deleted if row i is added to I.

Step 1. 'Column Deletion'

Let $\underline{RC} = RC_s$ be the smallest row cost (corresponding to row $s \in I'$).

(a) For each $j \in J$ such that $a_{sj} \neq 0$,

(i) Let $CP_j = CP_j + 1$.

(ii) If $CP_j = 1$ then delete j from J and for each $i \in I'$ such that $a_{ij} \neq 0$, update row costs letting $RC_i = RC_i - 1$.

(b) Move s from I' to I.

Step 2. 'Row-inclusion Penalties'

For each $i \in I'$, compute a row penalty

$$RP_i = \begin{cases} \sum_{\substack{a_{ij} \neq 0 \\ j \in J}} (CP_j + 1) & \text{if } CP_j < 0 \text{ for all } a_{ij} \neq 0, \\ \infty & \text{otherwise.} \end{cases}$$

Step 3. 'Row Addition'

Let $\underline{RP} = RP_s$ be the smallest row penalty (corresponding to row $s \in I'$).

If $\underline{RP} \leq 0$ then

(a) Move s from I' to I.

(b) For each $j \in J$ such that $a_{sj} \neq 0$

(i) Let $CP_j = CP_j + 1$.

(ii) If $CP_j = 0$ then for each $i \in I'$ such that $a_{ij} \neq 0$, update row costs letting $RC_i = RC_i + 1$.

(c) Go to Step 2.

Step 4. 'Incumbent Test'

If $|I|+|J|>|I_{R,C}|+|J_{R,C}|$ then let $I_{R,C}=I$ and $J_{R,C}=J$.

Step 5. 'Termination'

If $|I|<m$, then go to Step 1. Otherwise, print $I_{R,C}$, $J_{R,C}$ and STOP.

End of Algorithm GNRC

A stronger test, allowing preemptive termination, is possible at Step 5: If $|I|<m$ and $m+|J|>|I_{R,C}|+|J_{R,C}|$. However, the weaker test permits the exploration of a complete trajectory for $GN_{R,C}$ as discussed above.

Along the lines of UB_R, an easily computed upper bound on $M(GN_{R,C})$, denoted $UB_{R,C}$, was developed to check the accuracy of GNRC. Partition A as follows:

$$A=\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right].$$

Let z, z_{11} and z_{22} be the solutions to $M(GN_R)$ on A, A_{11} and A_{22}, respectively, and let UB_{11} be any simple upper bound on $M(GN_{R,C})$ for A_{11}. Then,

$$z \leqslant z_{11}+z_{22} \leqslant UB_{11}+z_{22}.$$

Algorithm UBRC computes $UB_{R,C}$ by iteratively applying the above statement, computing the simple bound UB_{11} and letting $A=A_{22}$ after each iteration. This is repeated until all columns of A_{22} have at most two nonzero elements in them at which point z_2 is equal to the number of rows plus the number of columns in A_{22}. $UB_{R,C}$ is then given by the sum of the UB_1 upper bounds found at each iteration plus z_{22} found at the last iteration. If A_{11} is selected such that it consists of single column and three rows, all with nonzero elements, then $UB_{11}=z_{11}=3$. Computational experience has indicated an effective rule for selecting the partition: among all columns in A having at least 3 nonzeros, select that column having the minimum number of nonzeros, and within that column select the first three rows with nonzeros in them. If k partitions are carried out before A_{22} becomes a GN matrix, it follows that:

$$UB_{R,C}=3k+|I_{22}|+|J_{22}|=3k+(|I|-3k)+(|J|-k)=|I|+|J|-k.$$

The last equality is used in computing $UB_{R,C}$.

Algorithm UBRC

Input: The LP coefficient matrix A.

Output: A value $UB_{R,C}$, an upper bound on $|I_{R,C}|+|J_{R,C}|$.

Step 0. 'Initialization'

Initialize:

(a) $I=\{1,2,\ldots,m\}$, and $UB_{R,C}=|I|+|J|$.

(b) For each column j, a column count

$$CC_j = \left(\sum_{\substack{a_{ij} \neq 0 \\ i \in I}} 1 \right).$$

Step 1. 'Iterative Partitioning'

Let $\underline{CC} = CC_s$ be the smallest column count greater than 2 (corresponding to column s).

If no such column exists, go to step 2. Else,

(a) Let $UB_{R,C} = UB_{R,C} - 1$.

(b) For exactly three $i \in I$ such that $a_{is} \neq 0$,

(i) Delete i from I.

(ii) For each j such that $a_{ij} \neq 0$, update column count letting $CC_j = CC_j - 1$.

(c) Repeat Step 1.

Step 2. 'Termination'

Print $UB_{R,C}$ and STOP.

End of Algorithm UBRC

4. Computational experience

The algorithms described in Section 3 have been implemented in FORTRAN using the X-System [5] as the host optimization package. Table 1 identifies twenty-three LP and mixed integer programming (MIP) problems which have been collected from various sources over the years. Some of these models are very well known, e.g., Dantzig's PILOT and the U.S. Department of Energy's PAD and PIES, and most of them were sent to us because of their difficulty, solution expense, or outright solution failure on commercial optimization systems. Table 1 shows problem dimensions excluding right-hand sides and objective functions. Computation times displayed in Tables 2-4 are compute-seconds, accurate to the precision shown, for FORTRAN IV H (Extended) with Optimize(2), run on IBM 3033AP under VM/CMS.

Algorithms GNRa and GNRd were used to identify GN_R rows with Algorithm UBR used to give an upper bound on the total number of such rows. To check accuracy, we attempted, within budget limitations, to solve exactly the integer linear programs for $M(\text{GN}_\text{R})$ in those cases where $|I_\text{R}| < UB_\text{R}$. (We were successful in all but one case, as seen. Times for solving the ILPs averaged 214.1 seconds for those problems solved.) Results for GNRa and GNRd, given in Table 2, are (a) the size of the optimal GN_R set found by the ILP, (b) the size of this set as a percentage of total problem rows m, (c) the size of the GN_R set found by GNR, (d) the size of this set as a percentage of the ILP optimum, and (e) the time required by the algorithm. For GNRd, the column labeled $|I_\text{R}|$ uses the notation $a:b$ where a is $|I_\text{R}|$ and b is the number of rows in I_R which were gained in the addition phase of

Table 1

LP/MIP problem set

Problem	Constraints	Variables	Nonzero Elements	Model
AIR	170	3 040	6 023	Physical Distribution
ALUMINUM	4 045	6 805	27 917	Econometric Production & Distribution
COAL	170	3 753	7 506	National Energy Planning
CUBIC1	657	3 074	15 894	Combinatorics Problem
CUBIC2	2 689	11 905	63 361	Bigger Combinatorics Problem
CUPS	360	618	1 341	Production Scheduling
ELEC	784	2 800	8 462	Energy Production & Consumption
FERT	605	9 024	40 484	Production & Distribution
FOAM	999	4 020	13 083	Production Scheduling
FOOD	4 010	14 409	23 332	Production, Distribution & Inventory Planning
GAS	788	5 541	31 020	Production Scheduling
JCAP	2 486	3 849	9 510	Production & Shipment Scheduling
LANG	1 235	1 425	22 028	Equipment & Manpower Scheduling
NETTING	89	190	388	International Currency Exchange
ODSAS	4 647	4 995	30 832	Manpower Planning
PAD	694	3 297	15 541	Energy Allocation, Distribution & Consumption
PAPER	2 868	5 348	23 746	Econometric National Production
PIES	662	3 011	13 376	Energy Production & Consumption
PILOT	974	2 172	12 927	Energy Development Planning
REFINE	5 220	5 994	40 207	Oil Refinery Model
STEEL	831	1 276	9 808	Econometric Production & Distribution
TRUCK	220	4 752	30 074	Fleet Dispatch (Set Cover)
WADDING	2 991	15 001	82 708	Multicommodity Prod. & Distribution Planning

the heuristic. Problems are weighted equally in computing average percentages in the 'totals' row of the table. Times listed do not include input or output.

All GNR variants perform quite well. The addition phase in GNRd did not often contribute a significant fraction of the GN rows found, but the additional rows found helped make both GNRd variants slightly better than either of the GNRa variants. The best algorithm on this problem set, GNRd (Senju & Toyoda), finds an average of 99.1% of the maximum GN_R set on those problems which we can solve exactly. The GN_R sets average 62.3% of the total problem rows on these same problems. GNR computation times are nominal compared with actual solution times of the seminal LPs and MIPs.

Results for UBR, given in Table 3, include (a) the size of the optimal GN_R set, (b) the upper bound, (c) the upper bound as a percentage of the ILP optimum, and (d) the time required to find the upper bound. For comparison, we include (e) Dobson's upper bound labeled 'UBD_R' and (f) that bound as a percentage of the ILP optimum. Table 3 also displays some properties of GN_R as found by GNRd, Senju and Toyoda. These properties include (g), the number of disjoint embedded

Table 2

Results for algorithms GNRa and GNRd

Problem	$M(GN_R)$		GNRd Senju & Toyoda			GNRd Dobson			GNRa Toyoda			GNRa Greedy		
	ILP Opt.	% m	$\lvert I_R \rvert$	% Opt.	Time	$\lvert I_R \rvert$	% Opt.	Time	$\lvert I_R \rvert$	% Opt.	Time	$\lvert I_R \rvert$	% Opt.	Time
AIR	170	100	170	100	0.0	170	100	0.0	170	100	0.1	170	100	0.0
ALUMINUM	2 198	54.3	2 175:13	99.0	9.2	2 174:16	100	9.2	2 194	99.8	7.3	2 179	99.1	7.3
COAL	170	100	170	100	0.0	170	100	0.0	170	100	0.0	170	100	0.0
CUBIC1	312	47.5	291:8	93.3	0.4	295:7	94.6	0.3	293	93.9	0.2	295	94.6	0.2
CUBIC2	1 264	47.0	1 191:25	94.2	5.1	1 177:19	93.1	4.9	1 192	94.3	3.0	1 195	94.5	2.9
CUPS	333	92.5	333	100	0.0	333	100	0.0	325	97.6	0.1	325	97.6	0.1
ELEC	520	66.3	520	100	0.3	520	100	0.3	520	100	0.4	518	100	0.3
FERT	572	94.5	572	100	0.2	572	100	0.2	562	98.3	0.3	562	98.3	0.2
FOAM	951	95.2	951	100	0.1	951	100	0.1	951	100	0.6	951	100	0.6
FOOD	3 716	92.7	3 716	100	1.8	3 716	100	1.8	3 709	99.8	9.1	3 710	99.8	9.1
GAS	73	9.3	73	100	2.6	73:25	100	1.1	73	100	0.1	73	100	0.1
JCAP	1 013	40.7	1 002:2	98.9	4.0	1 007.6	99.4	4.0	970	95.8	2.2	960	94.8	2.2
LANG	714	57.8	714	100	1.5	714	100	0.8	714	100	0.7	714	100	0.7
NETTING	72	80.9	72	100	0.0	72	100	0.0	71	98.6	0.0	71	98.6	0.0
ODSAS	1 498	32.2	1 490:95	99.5	16.5	1 446:61	96.5	16.1	1 498	100	6.7	1 463	97.7	6.5
PAD	122	17.6	122	100	1.2	122	100	0.4	122	100	0.1	122	100	0.1
PAPER	1 836	64.0	1 831:7	99.7	3.9	1 807:12	98.4	3.9	1 804	98.3	4.0	1 795	97.8	4.0
PIES	288	43.5	288	100	0.4	285	99.0	0.3	284	98.6	0.2	284	98.6	0.2
PILOT	470	48.3	462:1	98.3	0.7	459:5	97.7	0.6	459	97.7	0.4	459	97.7	0.4
REFINE	3 128	59.9	3 110:1	99.4	14.1	3 109:5	99.4	13.6	3 085	98.6	12.9	3 070	98.3	12.8
STEEL	431	51.9	419:1	97.2	0.5	421:2	97.7	0.4	425	98.6	0.3	424	98.3	0.3
TRUCK	NA	NA	70:1	NA	0.2	70:2	NA	0.2	68	NA	0.0	60	NA	0.0
WADDING	2 211	73.9	2 208:1	99.9	4.7	2 208:1	99.9	3.4	2 182	98.7	4.8	2 152	97.3	4.7
Totals	22 062	62.3	21 950	99.1	67.4	21 871	98.9	61.6	21 841	98.6	53.5	21 722	98.3	52.7

NA indicates IP solution not available. (LP optimum is 85.)

Table 3
GNR features

Problem	$M(GN_R)$	Algorithm UBR			Dobson Bound		Embedded GN_R components				
	ILP Opt.	UB_R	% Opt.	Time	UBD_R	% Opt.	Total	Largest $(m+n)$	Smallest $(m+n)$	Null Cols.	Sing. Cols.
AIR	170	170	100	0.0	170	100	1	170+3 040	—	0	57
ALUMINUM	2 198	2 214	100.7	1.7	3 798	172.8	145	1 118+3 431	1+1	0	1 234
COAL	170	170	100	0.0	170	100	1	170+3 753	—	0	0
CUBIC1	312	324	103.8	0.2	595	190.7	36	150+1 716	1+2	124	612
CUBIC2	1 264	1 332	105.4	2.7	2 479	196.1	149	562+6 064	1+6	353	2 488
CUPS	333	336	100	0.0	353	106.0	13	60+102	12+12	72	74
ELEC	520	524	100.9	0.2	705	135.6	14	74+408	2+16	18	174
FERT	572	572	100	0.1	600	104.9	1	572+9 024	—	0	1 757
FOAM	951	957	100.6	0.0	991	104.2	11	311+1 321	1+1	14	1 161
FOOD	3 716	3 720	100.1	0.1	3 939	106.0	75	1 785+7 147	1+4	522	6 989
GAS	73	74	101.4	0.1	682	934.2	11	53+4 714	1+2	336	5 018
JCAP	1 013	1 031	101.8	0.2	2 162	213.4	130	116+468	1+2	82	1 305
LANG	714	726	101.7	0.1	1 122	157.1	3	704+1 225	1+2	189	311
NETTING	72	72	100	0.0	84	116.7	17	20+80	2+1	23	990
ODSAS	1 498	1 510	100.8	2.3	4 181	279.1	115	701+2 403	1+4	507	1 663
PAD	122	122	100	0.0	558	457.5	3	82+1 354	8+33	1 730	1 179
PAPER	1 836	1 863	101.5	0.4	2 730	148.7	402	285+1 601	1+1	675	1 761
PIES	288	296	102.8	0.0	571	198.3	35	146+1 615	1+2	926	720
PILOT	470	490	104.3	0.1	887	188.7	78	177+533	1+1	618	624
REFINE	3 128	3 179	101.6	0.8	4 766	152.3	574	1 353+2 928	1+1	364	2 158
STEEL	431	458	106.3	0.1	763	177.0	95	180+541	1+1	248	548
TRUCK	NA	105	NA	0.1	197	NA	2	69+3 028	1+18	1 706	2 345
WADDING	2 211	2 222	100.5	0.2	2 866	129.6	3	969+4 169	1+1	4 414	5 032

GN components, (h) the largest and smallest components, (i) the number of null columns, and (j) the number of singleton columns. These properties are of interest since the structure of the embedded generalized network affects the solution techniques used in an LP factorization. For example, components consisting of single rows may be handled most efficiently without utilizing a complete generalized network code.

UB_R is surprisingly tight, averaging 101.4% of the true maximum, and computation times are nominal. Dobson's bound is poor, averaging 203.1% of the true maximum. The GN components found usually consist of a few large components and numerous small components.

Table 4 gives the results obtained by Algorithm GNRC and Algorithm UBRC. Since no ILP optimum is known for $M(GN_{R,C})$ in most cases, the items displayed differ from those items displayed in Tables 2 and 3. The results reported for Algorithm GNRC are (a) the size of the $GN_{R,C}$ structure found, (b) the time in seconds required to find the structure excluding input and output, (c) the size of the $GN_{R,C}$ as a

Table 4

$GN_{R,C}$ results

Problem	Algorithm GNRC $\lvert I_{R,C}\rvert+\lvert J_{R,C}\rvert$	Time	% $(m+n)$	% $UBLP_{R,C}$	Alg. UBRC % $UB_{R,C}$	Time	$\lvert I_R\rvert+n$	$\lvert J_C\rvert+m$
AIR	3 210	0.0	100	100	100	0.0	3 210	3 210
ALUMINUM	9 027	13.6	83.2	91.7	91.7	2.3	8 980	5 508
COAL	3 923	0.0	100	100	100	0.0	3 923	3 923
CUBIC1	3 365	0.6	90.2	99.4	94.8	0.4	3 365	659
CUBIC2	13 096	11.1	89.7	99.5	94.7	6.4	13 096	2 690
CUPS	951	0.0	97.2	100	99.7	0.0	951	713
ELEC	3 322	0.3	92.7	99.0	98.3	0.2	3 320	1 042
FERT	9 596	0.3	99.7	100	99.9	0.2	9 596	2 362
FOAM	4 971	0.1	99.0	100	99.7	0.1	4 971	1 044
FOOD	18 137	0.8	98.5	99.5	99.4	0.1	18 125	17 860
GAS	5 920	5.4	93.5	94.9	94.5	0.2	5 614	848
JCAP	5 822	5.5	91.9	97.7	99.8	0.2	4 851	5 718
LANG	2 139	1.1	80.4	97.8	90.2	0.2	2 139	1 905
NETTING	262	0.0	93.9	97.8	100	0.2	262	256
ODSAS	7 556	40.0	78.4	78.0	86.1	1.2	6 470	5 094
PAD	3 621	3.9	90.7	98.8	95.3	0.3	3 419	2 416
PAPER	7 388	4.6	89.9	95.9	96.2	0.9	7 179	4 905
PIES	3 313	0.9	90.2	99.5	94.8	0.2	3 299	2 241
PILOT	2 645	1.4	84.1	95.7	91.6	0.2	2 634	1 567
REFINE	9 326	19.3	83.2	93.8	92.4	2.3	9 104	7 729
STEEL	1 700	0.9	80.7	91.5	89.7	0.2	1 695	1 131
TRUCK	4 822	0.5	97.0	NA	98.3	0.3	4 822	220
WADDING	17 209	8.3	95.6	99.7	97.8	1.0	17 209	14 451
Totals	141 321	118.1	91.3%	96.8%	95.6	17.1	138 232	87 582

NA indicates not available.

percentage of the total constraint matrix, and (d) the percentage of the LP upper bound ($UBLP_{R,C}$) achieved by the algorithm. The results reported for Algorithm UBRC are (e) $|I_{R,C}|+|J_{R,C}|$ as a percentage of $UB_{R,C}$, and (f) the time required to obtain $UB_{R,C}$. For comparison, the last two columns of the table give the total number of rows and columns obtained for the GN_R and GN_C problems. These are the sizes of the embedded GN submatrices when restricted to row submatrices and column submatrices, respectively. Each problem is weighted equally to compute average percentages in the 'Totals' row.

GNRC performs very well, also. The algorithm finds a $GN_{R,C}$ structure whose size averages 91.3% of the size of the total constraint matrix. The size of the structure averages 96.8% of the LP upper bound on those problems for which the bound was obtained. (Times to obtain the LP bound averaged 315.8 seconds.) With respect to $UB_{R,C}$, the $GN_{R,C}$ set found averages 95.6%. Thus, the upper bound provided by algorithm UBRC is only slightly weaker, on average, than the LP upper bound. In addition, UBRC has more than a 400 to 1 computational speed advantage over the LP upper bound making it very attractive.

Additional computational studies have been performed to investigate the structures which GNR and GNRC obtain. Figure 1 summarizes this work for ELEC, JCAP, PAD, PIES and PILOT. The outer rectangle represents, to scale, the constraint matrix for each problem. The area above the dashed line represents the GN_R set found by GNRd, Senju and Toyoda. Within this area are indicated the connected components found by a simple connectivity algorithm. As indicated previously in Table 3, a few large components are typically found together with numerous small components. The area to the left of the vertical line represents the GN_C set. The irregular lines trace the trajectories of the $GN_{R,C}$ structures found by GNRC, ranging from GN_R on the right to GN_C at the lower left. From any point on this trajectory, all rows and columns above and to the left form a GN set. The circle indicates the largest $GN_{R,C}$ structure found on this trajectory.

5. Conclusion

Although GN_C identification is easy, GN_R and $GN_{R,C}$ identification is theoretically difficult. However, maximal, and often optimal GN_R and $GN_{R,C}$ substructures can be found in an LP constraint matrix using the heuristic algorithms developed here. In some problems, large GN_R structures can be found, while in other problems, it is necessary to remove some columns to find a large embedded $GN_{R,C}$ structure. Since execution time is modest for heuristic GN identification, our algorithms can be applied as a matter of course in general LPs to seek GN substructures. Evidence from the problem set indicates that this is well-advised if a GN-exploiting method is available: no members of the problem set were known, *a priori*, to contain significant GN structure and yet, in several cases, GN structure was predominant.

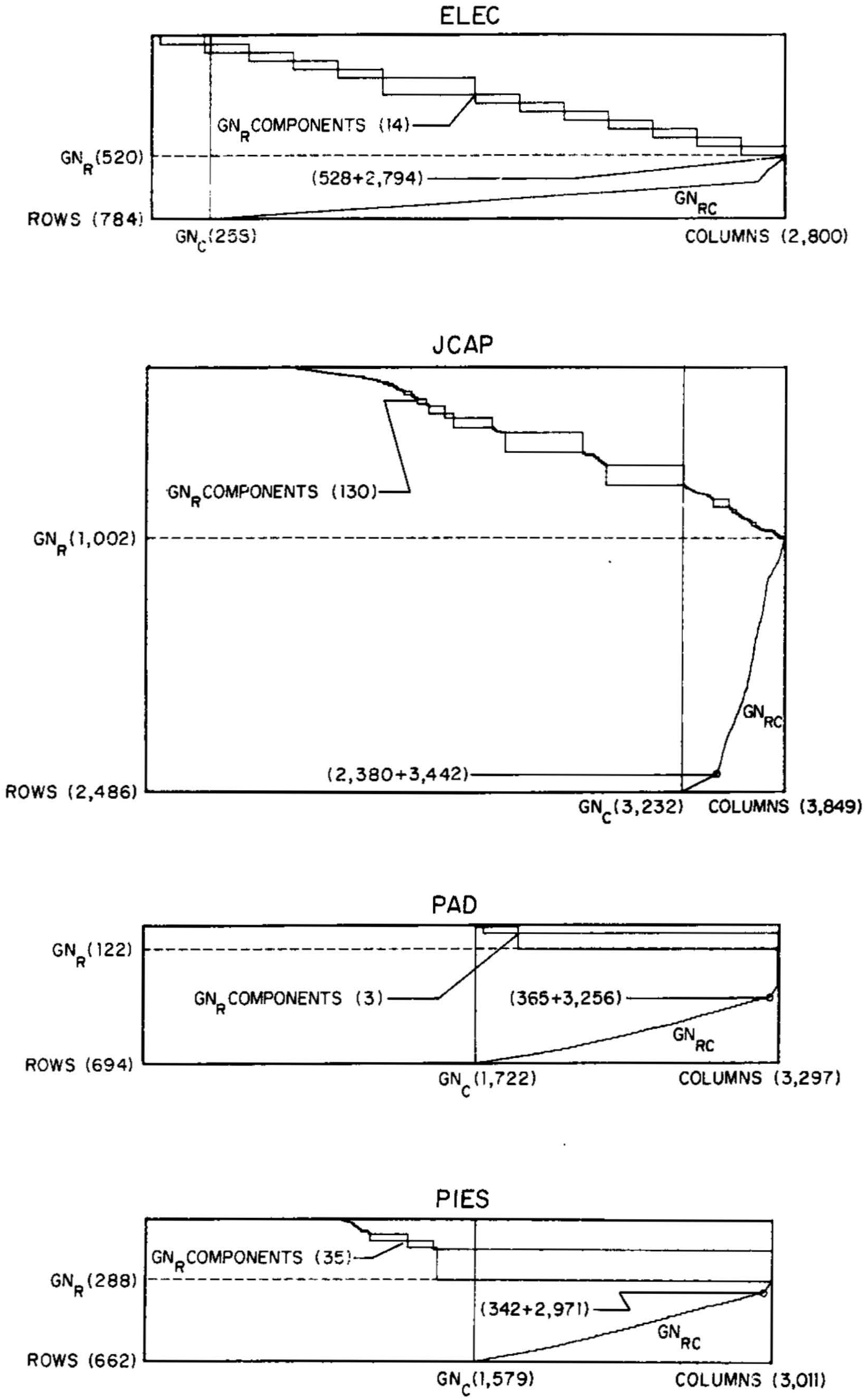

Fig. 1. Embedded generalized networks.

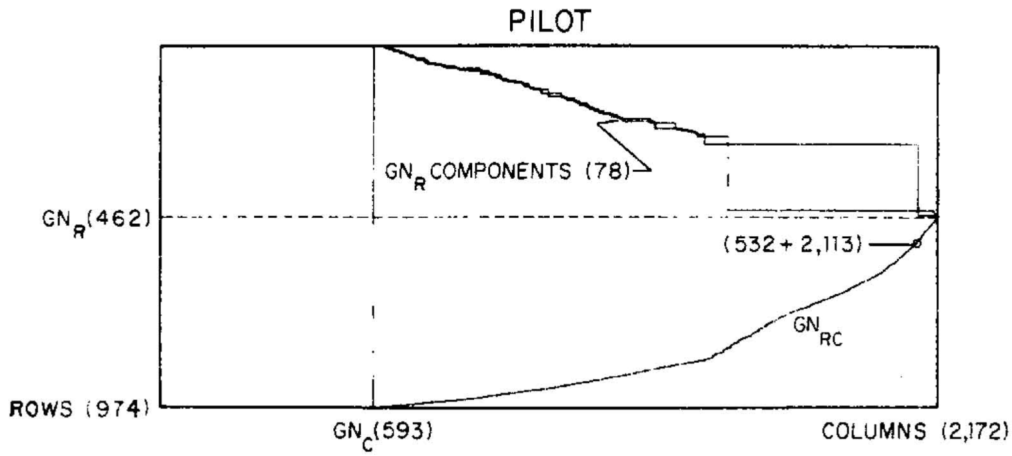

Fig. 1 (*cont.*).

Acknowledgements

We would not have begun this research and certainly not have sought the LP and ILP upper bounds without the support of Glenn Graves. Art Schoenstadt generously assisted in setting up the problem set for interactive experiments.

References

[1] J. Bartholdi, "A good submatrix is hard to find", *Operations Research Letters* 1 (1982) 190–193.

[2] R. Bixby and W. Cunningham, "Converting linear programs to network problems", *Mathematics of Operations Research* 5 (1980) 321–357.

[3] G. Bradley, G. Brown and G. Graves, "Implementation of large-scale primal transshipment algorithms", *Management Science* 24 (1977) 1–34.

[4] A. Brearley, G. Mitra and H. Williams, "Analysis of mathematical programming problems prior to applying the simplex algorithm", *Mathematical Programming* 8 (1975) 54–83.

[5] G. Brown and G. Graves, "Design and implementation of a large scale (mixed integer, non-linear) optimization system", presented at ORSA/TIMS Conference, Las Vegas, NV, November 1975.

[6] G. Brown and R. McBride, "Solving generalized networks", *Management Science* 30 (1984) 1497–1523.

[7] G. Brown and D. Thomen, "Automatic identification of generalized upper bounds in large-scale optimization models", *Management Science* 26 (1980) 1166–1184.

[8] G. Brown and W. Wright, "Automatic identification of embedded network rows in large-scale optimization models", *Mathematical Programming* 29 (1984) 41–46.

[9] G. Dobson, "Worst-case analysis of greedy heuristics for integer programming with nonnegative data", *Mathematics of Operations Research* 7 (1982) 515–531.

[10] M. Garey and D. Johnson, *Computers and intractability: A guide to the theory of NP-completeness* (W.H. Freeman, San Francisco, CA, 1978).

[11] F. Glover, J. Hultz, D. Klingman and J. Stutz, "Generalized networks: A fundamental computer-based planning tool", *Management Science* 24 (1978) 1209–1220.

[12] F. Glover and D. Klingman, "The simplex son algorithm for LP/embedded network problems", *Mathematical Programming Study* 15 (1981) 148–176.

[13] G. Graves and R. McBride, "The factorization approach to large-scale linear programming", *Mathematical Programming* 10 (1976) 91–110.

[14] G. Graves and T. Van Roy, "Decomposition for large-scale linear and mixed integer programming", Technical Report, University of California at Los Angeles (Los Angeles, CA, November 1979).

[15] R. Karp, "Reducibility among combinatorial problems", in: R. Miller and J. Thatcher, eds., *Complexity of computer computations* (Plenum Press, New York and London, 1972) pp. 85–103.

[16] S. Lin, "Computer solutions of the traveling salesman problem", *Bell System Technical Journal* 44 (1965) 2245–2269.

[17] S. Lin and B. Kernighan, "An effective heuristic for the traveling salesman problem", *Operations Research* 21 (1973) 498–516.

[18] R. McBride, "Solving generalized network problems with side constraints", Working Paper, FBE Department, School of Business Administration, University of Southern California (Los Angeles, CA, September 1981).

[19] R. McBride, "Solving embedded generalized network problems", to appear, *European Journal of Operations Research* (1985). Also, Working Paper, FBE Department, School of Business Administration, University of Southern California (Los Angeles, CA, October 1982).

[20] J. Musalem, "Converting linear models to network models", Ph.D. Dissertation, University of California at Los Angeles (Los Angeles, CA, January 1980).

[21] 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) pp. 389–395.

[22] S. Senju and Y. Toyoda, "An approach to linear programming with 0–1 variables", *Management Science* 15 (1968) B196–B207.

[23] Y. Toyoda, "A simplified algorithm for obtaining approximate solutions to zero-one programming problems", *Management Science* 21 (1975) 1417–1427.

[24] M. Yannakakis, "Node-deletion problems on bipartite graphs", *SIAM Journal on Computing* 10 (1981) 310–327.