The Null Space Problem II: Algorithms*

Thomas F. Coleman[1]
Alex Pothen[2]

TR 86-747
April 1986

Department of Computer Science
Cornell University
Ithaca, NY 14853

* Also Available as Technical Report CS-86-09, Computer Science Department, The Pennsylvania State University, University Park, PA 16802.

[1] Computer Science Department, Cornell University, Ithaca, NY 14853.
[2] Computer Science Department, The Pennsylvania State University, University Park, PA 16802.

The Null Space Problem II: Algorithms

Thomas F. Coleman
Department of Computer Science
Cornell University
Ithaca, NY, 14853

and

Alex Pothen
Department of Computer Science
The Pennsylvania State University
University Park, PA, 16802

Abstract

The Null Space Problem is that of finding a sparsest basis for the null space (*null basis*) of a $t \times n$ matrix of rank t. This problem was shown to be NP-hard in Coleman and Pothen (1985). In this paper we develop heuristic algorithms to find sparse null bases. These algorithms have two phases: In the first combinatorial phase, a minimal dependent set of columns is identified by finding a matching in the bipartite graph of the matrix. In the second numerical phase, a null vector is computed from this dependent set.

We describe an implementation of our algorithms and provide computational results on several large sparse constraint matrices from linear programs. One of our algorithms compares favorably with previously reported algorithms in sparsity of computed null bases and in running times. Unlike the latter, our algorithm does not require any intermediate dense matrix storage. This advantage should make our algorithm an attractive candidate for large sparse null basis computations.

A matching based algorithm is designed to find orthogonal null bases, but we present some theoretical evidence that such bases are unlikely to be sparse. Finally, we show how sparsest orthogonal null bases may be found for an n-vector and a $t \times n$ dense matrix by a divide and conquer strategy. The algorithm for a dense matrix is suited for implementation on a parallel machine architecture.

Key Words. null basis, null space, sparse matrix, bipartite graph, matching, orthogonal matrix, linear programming, nonlinear programming

AMS(MOS) subject classifications. 05, 15, 49, 65, 68

This research was supported in part by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under contract DE-AC02-83-ER10369, and a Cornell fellowship to the second author. A preliminary version of this work was presented at the SIAM Conference on Applied Linear Algebra at Rayleigh, NC, in April 1985.

Table of Contents

1. Introduction

Currently successive quadratic programming is the most popular method to solve constrained nonlinear optimization problems. The quadratic programming subproblems are often solved by numerically stable null space algorithms. Thus designing efficient null space algorithms for large scale optimization problems is an area of intense research effort at present. One concern is that these algorithms require a sparse representation of the null space of the constraint matrix.

Let A be a $t \times n$ matrix of rank t. The Null Space Problem (NSP) (Coleman and Pothen (1985)) is to find a basis N, with the fewest nonzeros, for the null space of A. For brevity, a basis for the null space will be called a *null basis*, and a column of a null basis will be called a *null vector*.

Two representations for the null basis N have been used so far in optimization algorithms. Wolfe (1962) proposed permuting the columns of A to obtain a $t \times t$ nonsingular matrix M such that $A = (M\,U)$, so that

$$N = \begin{pmatrix} B \\ I_{n-t} \end{pmatrix},$$

where $B \equiv -M^{-1}U$, and I_{n-t} is the $(n-t)$-dimensional identity matrix. We will call such a basis a *fundamental* null basis. An *explicit representation* of N is one in which the nonzeros in N are stored. In practice, N is represented *implicitly* by storing the LU factors of M, and a matrix-vector product such as Np is computed by solving a system of equations involving M. In the second representation, an LQ factorization of A is computed, and the last $n-t$ columns of Q form an orthogonal null basis for A. This scheme is impractical for large scale problems since Q is likely to be quite dense.

A context in which an explicit representation of the null basis is required concerns the optimization of a nonlinear function subject to linear constraints. A null space method demands the matrix $N^T HN$, where H is the current Hessian or an approximation to it. If a subroutine to compute the gradient of the objective function is available, then it is possible to obtain a good approximation to HN by $n-t$ extra gradient evaluations (Gill, Murray, and Wright (1981)), provided N is explicit. If N is explicit, and HN is sparse, then, in general, many fewer gradient evaluations will be needed if a sparse finite difference scheme is used (Coleman and Moré (1983), (1984)).

A second context arises from the recent work of Goldfarb and Mehrotra (1985) and Shanno and Marsten (1985) which extends Karmarkar's algorithm for linear programming. The crucial computational steps in these works are the solution of large sparse linear least squares problems of the form

$$DNw = b,$$

where D is a diagonal matrix, and N is a null basis of the constraint matrix. Both groups suggest solving the linear systems using pre-conditioned conjugate gradients; however, a host of pre-conditioning strategies is lost if N is not explicit. For example, diagonal, incomplete Cholesky, and chordal (Coleman (1986)) pre-conditioners, all require an explicit null basis N. (Thapa (1984) discusses a variety of pre-conditioners available for optimization problems: most require explicit matrices.)

Hence, for the rest of this paper, we restrict ourselves to the study of sparse explicit representations of null bases.

Previous work by others. Recently much work has been done on computing sparse null bases. The 'turnback' method for computing a sparse banded null basis for sparse banded equilibrium matrices for structural analysis was proposed by Topcu (1979). Kaneko, Lawo, and Thierauf (1982) interpreted this algorithm from a matrix factorization point of view. Berry, Heath, Kaneko, Lawo, Plemmons, and Ward (1984) refined this algorithm, developed software which used sparse banded data structures, and tested it on several structural engineering problems. Berry and Plemmons (1985) have implemented this algorithm on a HEP multiprocessor.

The turnback algorithm computes a QR factorization of A to identify a set of $(n-t)$ start columns. These are columns which are identified as linearly dependent in the factorization. Hence there is a null vector containing a start column and columns numbered lower than it in the matrix. Each null vector is computed by an algorithm which maintains a set of active columns, initially containing only a start column. Lower numbered columns are added to the active set, one by one, and a QR factorization of the active set is maintained. When the active set becomes dependent, the columns correspond to the nonzero components of a null vector. If the dependence involves the start column, the null vector is accepted. If not, the dependent column is rejected from the active set, and the process continued.

Null bases obtained by turnback are not fundamental; they have an embedded upper triangular matrix U_{n-t} of dimension $(n-t)$ with nonzero diagonal elements. Thus

$$N = \begin{pmatrix} B \\ U_{n-t} \end{pmatrix},$$

and we call such bases *triangular* null bases.

Gilbert and Heath (1986) have implemented several algorithms to compute sparse null bases. One of these is the turnback algorithm using general sparse data structures from Sparspak (George and Liu (1981)). Another is a matching based algorithm that computes triangular bases; we discuss this algorithm in section 4 of this paper.

Previous work by the authors. The Null Space Problem (NSP) was formulated in Coleman and Pothen (1985). We briefly summarize definitions and major results from that paper that will be useful here.

A null vector of A can be obtained from a linearly dependent set of columns. We call such a set a *dependent set*. The coefficients of the linear combination correspond to the values of the nonzero components of the null vector. A minimal dependent set of columns of A is a *circuit*. We proved that only null vectors that correspond to circuits could be columns in a sparsest null basis. Sparsest null bases were characterized by a greedy algorithm that augmented a partial basis by a sparsest null vector independent of those previously chosen. Despite this result, finding a sparsest null basis is computationally an intractable problem since it is NP-hard. Computing a sparsest fundamental null basis is also an NP-hard problem.

We addressed the question if sparsest null bases could be characterized to have some particular zero-nonzero structure (*structure*). It is known that a sparsest null basis may not be fundamental. We showed that a set of k vectors is linearly independent for all possible numeric values of its nonzeros if and only if it has an embedded upper triangular matrix of dimension k. Yet we do not know if we can always restrict a sparsest basis to be triangular. Nevertheless, restriction of the structures to fundamental and triangular bases makes it easy to ensure linear independence of the null vectors.

Outline of this paper. In this paper we report on the design and implementation of algorithms to compute fundamental and triangular null bases. These algorithms have two phases: in the first combinatorial phase, we identify the nonzero positions in the null vectors. The nonzeros in each null vector corresponds to columns of A in a circuit. In a second numeric phase, numeric values of the nonzeros are computed.

In section 2, we design an algorithm that finds a dependent set from a maximum matching of A. If the matrix satisfies a non-degeneracy assumption, then this dependent set is a circuit. We also show that any circuit of A can be found from an appropriate matching. The matching theory needed to understand this paper is introduced as needed in this section.

We show how the circuit algorithm can be used repeatedly to find a fundamental null basis in section 3. We attempt to compute sparse bases heuristically by assigning weights to the columns and choosing a a column of minimum weight to match to an unmatched row. We report on an implementation and on our computational results.

Section 4 describes an algorithm to compute triangular null bases. Here a circuit is obtained from a start column by choosing columns to add to it so that the circuit remains sparse. The algorithm is guided in its choice of columns by simultaneously constructing a matching as the columns are being added. An implementation is described and the sparsity of fundamental and triangular bases are compared. We compare this algorithm with the algorithms of Gilbert and Heath (1986).

Computing orthogonal null bases is addressed in section 5. We show that the circuit algorithm can be used to compute orthogonal bases. We characterize the columns in a circuit by means of a block lower triangular decomposition of a matrix. This characterization is exploited to study how sparse orthogonal bases are likely to be.

In section 6, we show that sparsest orthogonal null bases can be computed by Givens rotations for two special cases: an n-vector and a dense $t \times n$ matrix. A divide and conquer strategy that recursively splits the problem into two equally sized subproblems is shown to be optimal.

Section 7 contains a summary of our work.

By convention, a term is in *slanted* font when it is being defined.

2. A Circuit Algorithm

We compute a sparse null vector in two phases. In the first, we identify a dependent set using the structure of the matrix. In the second, the nonzero coefficients of a null vector are computed by solving a system of equations. We proceed to introduce the matching theory needed to design a circuit algorithm.

The *bipartite graph* $G(A)$ of the matrix A has a row vertex corresponding to each row of A, and a column vertex corresponding to each column of A. An edge joins a row vertex to a column vertex if and only if the corresponding matrix element is nonzero. The structure of a matrix and its bipartite graph are shown in Figure 2.1. The symbols '$\times$', and '$\otimes$' denote nonzeros; the rest are zeros.

A *matching* in A is a set of nonzeros of A such that no two elements in the set are chosen from the same column or the same row. A matching of A corresponds in $G(A)$ to a set of edges no two of which are incident on a common vertex. A matching in the matrix of Figure 2.1 is shown by circled nonzero elements; in the bipartite graph the edges in the matching are drawn with thick lines. A vertex is *matched* if it is an endpoint of an edge in a matching. A vertex that is not matched is *unmatched.*

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} \times & \times & 0 & \otimes & 0 & 0 \\ \times & 0 & \otimes & \times & 0 & \times \\ 0 & 0 & 0 & 0 & \otimes & \times \end{pmatrix} \end{array}$$

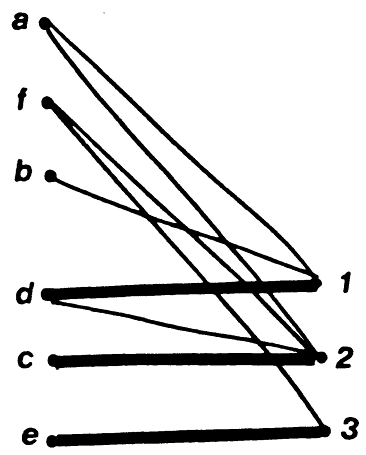

Figure 2.1.

The structure of a matrix, its bipartite graph, and a matching.

The matching $\mathcal{M}$ in the figure has maximum cardinality, and hence is a *maximum matching* of A. The *matching number*, $m(A)$, is the cardinality of a maximum matching of A. A matching in which all the rows are matched is a *complete* (row-perfect) matching in A. A matching in which all rows and columns are matched is a *perfect* matching.

There exist several polynomial time algorithms to find maximum matchings in bipartite graphs. Let τ denote the number of nonzeros in A. The theoretically fastest known algorithm is due to Hopcroft and Karp (1973), and has time complexity $O(t^{1/2}\tau)$. Duff (1981) prefers an $O(t\tau)$ algorithm which he finds is faster in practice. Good discussions of matching algorithms may be found in Papdimitriou and Steiglitz (1982), and Lawler (1976).

The matrix A has the *Hall property* (HP) if every subset of its rows has nonzeros in at least as many columns.

Proposition 2.1. *(Philip Hall) A has a complete matching if and only if it has the Hall property.*

Proposition 2.2. *The matching number of a matrix is greater than or equal to its rank.*

Thus a matrix with rank t has a complete matching. A stronger condition on A is the Strong Hall property (SHP). The matrix A has the *Strong Hall property* if every subset of $0 < k < n$ rows has nonzeros in at least $k+1$ columns. (Thus when $t < n$, every set of $k \leq t$ rows has nonzeros in $k+1$ columns, and when $t = n$, every set of $k < n$ rows has nonzeros in $k+1$ columns.) The terms (HP) and (SHP) are due to Coleman, Edenbrandt, and Gilbert (1983).

A complete matching $\mathcal{M}$ of A partitions the columns of A into two sets: M, the set of matched columns, and U, the set of unmatched columns. In Figure 2.1, $M = \{c, d, e\}$ and $U = \{a, f, b\}$. We now show that for a column $u \in U$, we can construct a circuit of A containing u by an 'alternating path algorithm'.

A *path* in a graph is a sequence of distinct vertices $v_1, \ldots, v_k$, where (v_{i-1}, v_i) is an edge of the graph, for $1 < i \leq k$. An $\mathcal{M}$-*alternating path* is a path whose edges are alternately chosen from the matching $\mathcal{M}$ and outside $\mathcal{M}$. In Figure 3.1 the sequence of edges $(b,1)$, $(1,d)$, $(d,2)$, $(2,c)$ is an $\mathcal{M}$-alternating path in A. We say that c and d are reachable from b by $\mathcal{M}$-alternating paths, and indicate this by $b \xrightarrow{M} c$ and $b \xrightarrow{M} d$.

For $u \in U$, the following algorithm constructs a dependent set $n(u)$ containing u; this is a circuit if A has the weak Haar property (wHP). A matrix has the *weak Haar property* if every set of columns C satisfies $rank(C) = m(C)$. This assumption ensures that $n(u)$ will be a circuit for all 'general' numeric values of the columns of A. For a particular set of numeric values of the nonzeros of A, 'unexpected' numerical cancellations may occur, in which case the set $n(u)$ will contain a circuit. (The definitions of (HP), (SHP), and (wHP) are tabulated in Table 1 for easy reference.)

Circuit Algorithm. Given a matrix A with (wHP), a complete matching $\mathcal{M}$, and an unmatched column $u \in U$, this algorithm finds a circuit $n(u)$.

> Follow all $\mathcal{M}$-alternating paths from u,
> adding columns visited to $n(u)$. Thus
> $n(u) = u + \{ v \in M : u \xrightarrow{M} v \}$.

From Figure 2.1 it is easy to see that $n(a) = \{a, d, c\}$, $n(b) = \{b, d, c\}$, and $n(f) = \{f, c, e\}$. The set $n(u)$ can be constructed in $O(\tau)$ time by a depth first search.

Theorem 2.3. *The set of columns $n(u)$ is a circuit if A has (wHP).*

Proof: Let C be the set of columns in the dependent set $n(u)$, and let C have nonzeros only in the row set R. For ease of notation, denote by B the submatrix A_{RC}.

We first show that B has (SHP). Consider any subset S of k rows of B. S is matched in $\mathcal{M}$ to k columns, all of which are in C. If the unmatched column u has a nonzero in any of the rows in S, then S has nonzeros in at least $k+1$ columns.

Suppose that u has no nonzero in S. Since rows in S are reachable from u by $\mathcal{M}$-alternating paths, there must exist a column, matched to a row outside S, with a nonzero in S. Again, S has nonzeros in at least $k+1$ columns.

Let b be any column in B. Since B has (SHP), $B \setminus b$ has (HP). By Proposition 2.1 $B \setminus b$ has a complete matching of size $|R|$. Since A has (wHP), the rank of $B \setminus b$ is also $|R|$, and so the columns in $B \setminus b$ are independent. Since B is dependent, it follows that $n(u)$ is a circuit. ▮

Let $\widetilde{C}$ denote the submatrix of columns in $C \setminus u$ and rows in R, and let $\tilde{u}$ denote the components of u corresponding to rows in R. The coefficients of the null vector can be computed by solving

$$\widetilde{C}x = -\tilde{u},$$

and then choosing:

$$n(u)_i = \begin{cases} x_i & \text{if i corresponds to a column in } \widetilde{C}; \\ 1 & \text{if i corresponds to } u; \\ 0 & \text{otherwise.} \end{cases}$$

Suppose that $\widetilde{C}$ does not have (wHP). Since it has a perfect matching by construction, it is rank deficient. Hence it may not be possible to express $\tilde{u}$ as a linear combination of columns in $\widetilde{C}$. In this case, we say that *the dependence in $n(u)$ does not involve u*. This situation is detected by a nonzero (significantly higher than machine precision) residual in the system of equations. Since $\widetilde{C}$ does not have full rank, it is possible to identify a dependent column, say m, from its numeric factorization. A null vector $n(m)$, with a nonzero component corresponding to m, can be computed from the independent columns in $\widetilde{C}$.

Corollary 2.4. *If $n(u)$ does not have (wHP), it contains a circuit which can be identified by a numeric factorization.* ▮

We now prove that the converse of Theorem 2.3 is true.

Theorem 2.5. *Every circuit of a matrix A with (wHP) can be constructed by the Circuit Algorithm from some maximum matching $\mathcal{M}$ of A.*

Proof: Let C be the set of columns in a circuit, and let R and B be as in the proof of Theorem 2.3. Denote $|C|$ by c, and distinguish any one column of B as u. We claim $B \setminus u$ has (HP).

Suppose not. Then R has a subset of k rows adjacent to fewer than k columns, for some k. The columns and rows of the submatrix B can be permuted to the structure in Figure 2.2.

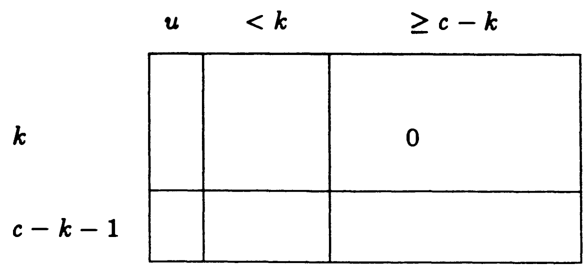

Figure 2.2.
The submatrix B.

The submatrix B then contains a dependent set of columns of size $c-k$, violating the minimality of C. Hence $B \setminus u$ has (HP), and by Proposition 2.1, it has a complete matching $\mathcal{M}_1$. Partition A as shown in Figure 2.3.

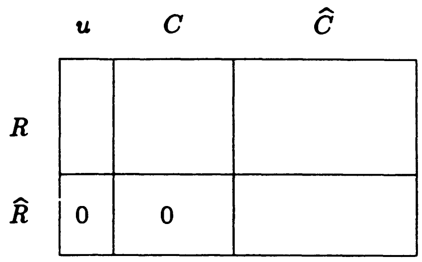

Figure 2.3.
A partition of A.

In any matching of A, the row set $\widehat{R}$ can match only to the column set $\widehat{C}$. Let $\mathcal{M}_2$ be a maximum matching of the submatrix $A_{\widehat{R}\widehat{C}}$. The required maximum matching is $\mathcal{M}_1 \cup \mathcal{M}_2$. ∎

3. Fundamental Null Bases

We now develop an algorithm to compute fundamental null bases using the Circuit Algorithm. The resulting algorithm will be modified later to reduce its complexity.

Algorithm 1.

1. [initialize] Let N be the empty set.
2. [match]
   Find a complete matching $\mathcal{M}$ of A;
   partition the columns: $A = (\,M \;\; U\,)$.
3. [construct basis]
   for $u \in U \;\rightarrow$
   construct $n(u)$ by the Circuit Algorithm;
   solve for the coefficients in $n(u)$;
   Augment the null basis N with the computed null vector.
   rof.

When A has (wHP), by Theorem 2.3, each set $n(u)$ is a circuit. Further, since an unmatched column u is contained only in the circuit $n(u)$ by construction, N is a fundamental null basis. Thus the algorithm is correct in this case.

Step 3 of Algorithm 1 can be modified to reduce its complexity. The fundamental null basis computed by Algorithm 1 from the partition $A = (M\,U)$ has the structure

$$N = \begin{pmatrix} B \\ I_{n-t} \end{pmatrix},$$

with $B \equiv -M^{-1}U$. Thus when M has full rank, the coefficients of each null vector $n(u)$ can be obtained by solving a system of the form $Mx = -u$. Thus we do not need to identify columns in $n(u)$ by following alternating paths. Also, the $(n-t)$ matrix factorizations in step 3 can be replaced by one.

This observation shows that when A does not have (wHP), the algorithm can fail to compute all the $(n-t)$ linearly independent null vectors in a basis. Corresponding to each fundamental basis, there is an associated partition of the columns of A into M and U. Since B satisfies the equation $MB = -U$, when M does not have full rank, it may not be possible to express a column u as a linear combination of the columns in M.

Thus when A does not have (wHP), a fundamental basis can be computed only when M has full rank. Hence we choose M by a matching, but ensure that M has full rank when we factor it to compute the null vectors. If it is rank-deficient, we reject the dependent columns in M from the matching, and find a new maximum matching. This strategy will ensure correctness; and will always succeed when A has full row rank.

For some of the problems reported here, the submatrix M chosen by a matching was indeed rank deficient. The number of dependent columns was almost always equal to one or two out of a few hundred columns; the largest we observed was five.

Details of implementation. Since computing a sparsest fundamental null basis is NP-hard, heuristic strategies have to be employed to find sparse bases. Our strategy is to assign costs to the columns of A, and to choose a column of minimum cost to match to a row.

The cost of a column c is the number of nonzeros in it. To justify this, observe that for a matrix with (wHP), the number of nonzero rows in a circuit is one smaller than its number of columns. Thus a sparse circuit has few nonzero rows. Hence the cost of a column c indicates that any circuit containing c must have at least this number of additional matched columns in it. The cost of a matching is the sum of the costs of the matched columns. Ties were broken in favor of lower numbered columns.

Our weighted maximum matching routine is derived from MC21A, Duff's algorithm for finding a maximum matching in a matrix (Duff (1977)). A maximum matching is obtained by matching the rows one by one. At one step of the matching algorithm, we search for a column to match to an unmatched row. From the given row, a depth first search is performed through alternating paths to visit every unmatched column that could be reached by such a path. The cheapest of these columns is chosen.

The solution of the linear systems to compute the coefficients of the null vector is accomplished by using the LUSOL package of Gill, Murray, Saunders, and Wright (1986). This package is presently a part of MINOS (Murtagh and Saunders (1983)). The LUSOL routines draw on the work of Reid (1976), (1982) on sparse LU factorizations of unsymmetric matrices. Gaussian elimination with row and column pivoting is performed such that $M = LU$, where the matrix PLP^t is lower triangular, PUQ is upper triangular, and P, Q are permutation matrices. Markowitz's criterion is used to select the pivot element, subject to a bound on the size of elements in L for numerical stability. Parameters in LUSOL were set at their default values.

The matching algorithm has complexity $O(t\tau)$; Duff (1981) reports an $O(t) + O(\tau)$ experimental behavior. Since the dimension of M is t, the cost of factoring it in step 3 could be $O(t^3)$, and the $(n-t)$ backsolves could cost $(n-t)t^2$ operations. However, since M is sparse, a more realistic cost should be about $O(t^2)$ for the factorization and $O((n-t)t)$ for the backsolves.

Storage requirements of the algorithm are dominated by the storage required for A and the null basis N. The matrix A is stored in Sparspak column oriented data structures (George and Liu (1981)). Nonzeros and row indices are stored in column major order. For the use of matching routines, column indices of the nonzeros are stored in row major order. These require a double precision array of length τ, two integer arrays of length τ, and integer arrays of length $n+1$ and $t+1$. Let $\tau(M)$ denote the number of nonzeros in M; this is smaller than τ. LUSOL needs the submatrix M stored as an element list with parallel integer arrays for row and column indices. The factorization is stored in the data structure for M, for which a minimum length of $\tau(M)+4t$ is recommended. The null basis N is stored in Sparspak column oriented data structures of length equal to the number of nonzeros in the basis.

Results. We implemented Algorithm 1 in FORTRAN 77; our experimental code, *BASIS*, is structured and modular, and we believe it represents a careful and efficient implementation. The program was run on Penn State's VAX 11\780 (with attached floating point accelerator) under Berkeley 4.2 Unix. The f77 compiler was used to compile the code.

Constraint matrices from linear programming problems were used for tests, and are shown in Table 2. The first, *murty*, was taken from Murty (1983), and all the others were supplied to us by Dr. Michael Saunders. Our results are tabulated in Table 3. The algorithm found bases comparable in sparsity to the input matrices for all problems except *brandy*, for which there was a four fold increase in density. This seems to be caused by the restriction to fundamental bases, as will be seen in the next section.

The total time (seconds) reported includes the time needed to find a maximum matching, compute the LU factors of M, and solve for the null vectors. The matchings were found quite fast, and the relatively larger times for the problems *brandy*, *capri*, and *etamacro* were caused by dependence in M which necessitated finding a second matching. This step can be speeded up if the current matching is updated instead of finding a new matching as we have done.

Both *Murty* and *Israel* had embedded identity matrices of dimension t; thus these null bases were anomalously easy to find. The times reported for these problems should therefore be considered low.

The time for the factorization phase was surprisingly low. This is due to the high sparsity in M as a result of the column selection strategy in the matching algorithm, and the efficient method for computing sparse factors via LUSOL. Solving for the coefficients accounted for most of the time needed by the algorithm; solving for each null vector took less than a tenth of a second, but the large number of null vectors caused the large time requirement. We conclude also that the numerical phase of the algorithm dominates the combinatorial phase in computational time required.

The numerical quality of each null vector was checked by computing the residual of the system of equations used to find a null vector. In all cases, this was below machine precision. Condition numbers were estimated for the the constraint matrices and null bases of the first eight problems in Table 2 by the LINPACK condition estimator. The estimates for the null bases were lower than the estimates for the constraint matrices, except for share1b; here the null basis had a condition estimate of approximately 10^6, about ten times that of the constraint matrix.

4. Triangular Null Bases

Since a triangular null basis has the structure

$$N = \begin{pmatrix} B \\ U_{n-t} \end{pmatrix},$$

if A is partitioned to conform to N as $A = (M\,S)$, then we have $B = -M^{-1}SU_{n-t}$. Then

$$N = \begin{pmatrix} -M^{-1}S \\ I_{n-t} \end{pmatrix} U_{n-t}.$$

Thus a triangular basis is obtained from a fundamental basis by postmultiplying by an upper triangular matrix. From matrix algebra alone, it is hard to see that triangular bases can be sparser than fundamental bases. The theme of this section is that judiciously constructed, they are.

We now describe an algorithm that computes a triangular null basis by matching methods. The diagonal elements of the triangular basis correspond to a set of $n - t$ *start columns* in A. The algorithm computes a circuit containing each start column. Linear independence of the set of circuits follows from the structure of N.

Throughout this section we do not assume that A has (wHP). The description of Algorithm 2 includes the remedial actions needed. By a *null vector* $n(u)$ we mean a null vector with a nonzero component corresponding to u.

Let $S = \{s_1, \ldots, s_{i-1}\}$ be a set of start columns for which null vectors $\{n(s_1), \ldots, n(s_{i-1})\}$ have been computed. Columns in S will not be used in any of the null vectors to be computed in the future. Algorithm 2 maintains the invariant that $A - S$ has a complete matching $\mathcal{M}$. Thus columns of A are partitioned into a set M of matched columns and a set U of unmatched columns, and $S \subseteq U$.

There is a great deal of freedom in how a dependent set $n(u)$ is constructed in Algorithm 2. As in Algorithm 1, we could employ the Circuit Algorithm to find $n(u)$ from the matching $\mathcal{M}$.

Algorithm 2. Given a complete matching $\mathcal{M}$ in a matrix A, and a partition into matched columns M and unmatched columns U, this algorithm computes a triangular null basis N. The set S is the set of columns which have already been used as start columns.

```
S := ∅;
while U ≠ S do
        Choose a column u ∈ U − S;
        Construct the dependent set n(u) from columns in A − S;
        Solve for the corresponding null vector;
        if dependence involves u
                then S := S + u; N := N + n(u);
                else find a column m involved in the dependence;
                   Let n(m) be the associated null vector;
                   S := S + m; N := N + n(m);
                   if m ∈ M then let r be the row matched to m;
                                 𝓜 := 𝓜 − (r, m);
                                 Augment 𝓜 by matching r
                                 to a column in A − S;
                                 Let v be the newly matched column;
                                 M := M − m + v; U := U + m − v;
                            fi
                fi
        od
```

Or, we could find $n(u)$ from *another* matching in $A - S$, with a view toward obtaining a sparse null vector. Indeed, later in this section we present a Modified Circuit Algorithm that, given u, *chooses* columns in $A - S$ to form $n(u)$ by simultaneously constructing a matching. Thus for each column u, a different matching in $A - S$ is constructed. This permits more intelligent choices in the column selection strategy to achieve sparsity. Even in this situation, it turns out that it is essential to maintain a complete matching $\mathcal{M}$ in $A - S$ to ensure the correctness of Algorithm 2.

Initially the complete matching $\mathcal{M}$ partitions the columns into M and U, and S is empty. For $u \in U - S$, a dependent set $n(u)$ is constructed by a matching algorithm from the columns in $A - S$. The corresponding null vector is computed as described in section 2. If u is involved in the dependence, then N is augmented with the vector $n(u)$, and u is added to the set S. Otherwise, we can identify a dependent column m, and a null vector $n(m)$ can be computed. The column m is added to the set S, and N augmented with the vector $n(m)$. If $m \in M$, then the row r matched to m in $\mathcal{M}$ has to be matched to another column in $A - S$ to maintain the invariant. This is accomplished by augmenting $\mathcal{M} - (r, m)$ to a complete matching, and updating the sets M and U.

It is easily seen that a triangular basis is obtained if the null vectors are arranged in the reverse order in which they are computed.

We now describe the algorithm that we use to find the dependent set $n(u)$. The Modified Circuit Algorithm is a variant of an algorithm to find circuits, proposed by Gilbert and Heath (1986), based on the matching theory developed in this paper. We first describe our version.

Modified Circuit Algorithm. Given a column u, this algorithm finds a dependent set $n(u)$. Here S is the set of start columns for which null vectors have been computed by Algorithm 2, C is the set of active columns, and R is the set of active rows.

$C := \{u\}$;
$R :=$ {rows in which u has nonzeros} ;
repeat
  Find an augmenting path from an unmatched active row r to an inactive column $c \in A - S$;
  Augment by adding r and c to the matching;
  $C := C + \{c\}$;
  $R := R +$ {inactive rows in which c has nonzeros} ;
until all active rows are matched;

This algorithm identifies an *active submatrix* formed from a set R of *active rows* and a set C of *active columns* such that A_{RC} is dependent. Initially, the only active column is the column u, and the active rows are the rows in which u has nonzeros. A queue of active rows is maintained by the algorithm. At each step, a column is chosen to match to a row in the queue. The column is added to the set of active columns, and inactive rows in which the column has nonzeros are made active and added to the queue. At termination the active rows are perfectly matched to the active columns (excluding u). If A has (wHP), by Theorem 2.3, the active columns form a circuit.

This algorithm will always succeed in finding a dependent set $n(u)$. The reason is that $A - S$ has a complete matching $\mathcal{M}$ by the invariant of Algorithm 2, and u is chosen to be an unmatched column. Thus there is at least one circuit $n(u)$ in $A - S$ (which could be obtained by the Circuit Algorithm). If there is no other circuit containing u, the Modified Circuit Algorithm could find this one.

We now prove that Algorithm 2 is correct. Since the size of S increases by one at each step, the algorithm must terminate. We must prove that the algorithm will find a start column and a null vector containing it at each step.

Theorem 4.1. *Let A be a matrix which may not have (wHP). Let $1 \le i \le n-t$, and $S = \{s_1, \ldots, s_{i-1}\}$ be a set of start columns for which null vectors $n(s_1)$, ..., $n(s_{i-1})$ have been computed. There exists a column $s_i \in A - S$ such that Algorithm 2 computes a null vector $n(s_i)$ from the columns in $A - S$.*

Proof by induction on i: Initially M is a set of columns in a complete matching of A, and S is the empty set. Since $A - S$ has a complete matching, a dependent set $n(u)$ can be constructed for any unmatched column u. If the dependence involves u then $s_1 := u$. Otherwise, let m be a column in $n(u)$ involved in the dependence. Now $s_1 := m$, and $n(s_1)$ can be computed numerically.

To maintain the invariant, there are two cases to consider.

Case 1: $m \in U - S$. Then M remains a completely matched set of columns in $A - m$.

Case 2: $m \in M$. Then from the proof of Theorem 2.3, the set of columns $n(u) - m$ is perfectly matched (in the matching found by the Modified Circuit Algorithm) to its set of nonzero rows, say R. Since $n(u) - m \subseteq A - m$, it is possible to match all rows in R to columns in $A - m$.

Let $\widehat{R}$ denote the rest of the rows of A. Then columns in $n(u)$ have zeros in the rows in $\widehat{R}$, and hence a row in $\widehat{R}$ cannot, in any matching, match to any of these columns. Thus columns matched in $\mathcal{M}$ to rows in $\widehat{R}$ are disjoint from columns in $n(u) - m$. It follows that all rows in this set can be matched to columns in $A - m$. Hence $A - m$ has a complete matching.

The inductive step proceeds similarly. We have a completely matched subset of columns M in $A - S$. An unmatched column in $U - S$ is chosen as the start column u, and since $A - S$ has a complete matching, a dependent set $n(u)$ can be constructed; if the dependence involves u, then $s_i := u$. Otherwise, we find the column s_i exactly as s_1 was found. ∎

Gilbert and Heath use a version of the Modified Circuit Algorithm as a component in their algorithm to find triangular bases. The major difference is that they maintain a QR factorization of the active submatrix as each column is being added. This helps them identify null vectors that could not be predicted purely by matchings, since they can terminate the algorithm when a numerical dependence is detected, even when all rows have not been matched. In many applications, e.g. structural analysis and network problems, such dependences abound, since the constraint matrix has additional 'structure'. The exploitation of these dependences means that the matching phase can run faster.

However, they observed that the QR factorization of the active submatrix was quite dense. Unfortunately, since columns are added to the submatrix in unpredictable order, ordering strategies could not be used to keep the factors sparse. Hence they stored the factorization as a dense matrix, with R in the upper triangle, and Householder transformations whose product is Q in the lower triangle.

We find a dependent set by matching methods alone, and then solve for the coefficients in the null vector by a sparse LU factorization of the submatrix A_{RC}. This submatrix is quite sparse; a sparse matrix factorization can be used to keep its LU factors sparse. Most important, a sparse matrix storage scheme can be used in the factorization. This enables us to avoid the storage problem that Gilbert and Heath have encountered with 'on the fly' QR factorizations.

Algorithm 2 differs from the matching based algorithm of Gilbert and Heath also in the strategy to select start columns. They use an initial QR factorization of A to identify a set of dependent columns, which is designated as the set of start columns. For each start column, they computed a null vector containing it, and any dependences not involving the start column were rejected to ensure that a basis was computed.

Details of implementation. The initial matching $\mathcal{M}$ in Algorithm 2 is chosen by the column weighting strategy in section 3. The column u is chosen to be a column with most nonzeros in $U - S$, since once this null vector is computed, u will not be used again. The column m is chosen to be a column in $U - S$ rather than in M, if possible, since this saves the $O(\tau)$ operations needed to update the matching. The column v is chosen to to be a column with fewest nonzeros in $U - S$ that can augment the matching.

In the Modified Circuit Algorithm, the column c is chosen to be an inactive column of minimum cost. Here, the cost of a column c is the number of nonzeros it has in the inactive rows. Our heuristic justification is that this number of additional unmatched rows are added to a circuit when c is matched to an active row. In case of ties, two different tie breaking strategies were tried: columns with fewer total nonzeros were favored in one, and in the other, columns with most total nonzeros. The first strategy worked better for almost all problems.

Each null vector is computed by Algorithm 2 from a perfect matching in a submatrix of A. Each submatrix can have t rows and $t+1$ columns, and hence the matching could cost $O(t\,\tau)$. The associated factorization could cost $O(t^3)$. The actual cost for each null vector should be lower since the number of rows in each submatrix should be small due to sparsity. Storage requirements for Algorithm 2 are similar to Algorithm 1.

Results. An implementation of our algorithm in FORTRAN 77 forms the second part of BASIS. We present our results (computed under the same conditions as for fundamental null bases) in Table 4.

The triangular null bases we obtained are consistently sparser than fundamental bases. The increase in sparsity is most spectacular for *brandy* where the triangular basis has about half the density of the latter. In all cases, the densities of the constraint matrices and triangular null bases are comparable.

The time reported is the time (in seconds) needed to find a basis, given the matrix stored in Sparspak data structures and an initial complete matching. Algorithm 2 requires about two the time needed by fi Algorithm 1. This is caused primarily by the $(n - t)$ matrix factorizations needed to compute the null vectors. The size of most of the matrices to be factored is quite small since the null vectors are sparse. Also, the LUSOL routines compute the sparse factorization efficiently. This explains why the increased time requirement is not greater.

The residuals in the system of equations from which null vectors are computed were always below machine precision. Condition numbers were estimated as before for the first eight problems in Table 2. These were all small, except for share1b and brandy ($cond(N) \approx 10^6$ for both); for the former the ratio $cond(N)/cond(A)$ was about ten, and for the latter about hundred.

We have also run Algorithm 2 on seven test problems shown in Table 5 from Gilbert and Heath (1986). Five of these are equilibrium matrices from structural optimization, and two, lp1 and lp2, are obtained from linear programs. In Table 6, we compare the results from Algorithm 2 with the reported results of Gilbert and Heath on GHM and GHT. Their code was executed on a reasonably similar setup to ours—a VAX 11\780 running the same operating system and on the same compiler. However, we need to be cautious about attaching too much significance to small differences in running times, since the algorithms are not being compared on the same machine.

The times (in seconds) reported by Gilbert and Heath exclude the time needed for column preordering and the initial QR factorization. Storage reported is the intermediate storage required to obtain a dependent set. For GHM and GHT, this is the maximum size of the dense matrix storage needed for the active submatrix. (Additionally, integer arrays are needed to keep track of the rows and columns in the submatrix; this is not included in the reported storage.) For Algorithm 2, this is the maximum number of nonzeros in the L and U factors of the dependent set. We have not included the storage required by the integer arrays needed for column and row indices of nonzeros.

Generally, Algorithm 2 and GHM outperform GHT, both in sparsity of the bases and in execution time. The difference in running times between the matching algorithms and the turnback algorithm is significant. The matching algorithms run about two to twenty times faster because they identify circuits combinatorially. The two matching algorithms are comparable in running times for the structural problems. But, on lp2, the larger linear program, GHM needed more than twice the time of Algorithm 2, and GHT about twenty. We have not implemented GHM or GHT, and do not have results for these algorithms on the linear programs in Table 2. But we should expect Algorithm 2 to have an even greater edge over GHM and GHT on the larger problems in this set.

The intermediate storage required is the criterion where the three algorithms differ greatly. To solve lp2, GHT has to store an active submatrix of size $13,000$, and GHM, a submatrix of size $8,000$. The intermediate storage needed by Algorithm 2 is 700, much less than the number of nonzeros in A. This behavior is typical over the entire set of problems. Both GHM and GHT are limited by the intermediate storage required in the size of problems that can be solved. Algorithm 2 is not so limited, and should be preferred in solving large null basis problems.

GHM has a small edge over Algorithm 2 in sparsity for the structural problems. For some problems, this may be attributed to column pre-orderings employed. However, this advantage is not surprising to us, since our algorithm works with a flawed model of a circuit for the structural problems, as we explain in section 7.

5. Orthogonal Null Bases

We now show how the Circuit Algorithm can be used to compute null bases with orthogonal columns. Let n_1 be a null vector of A computed by the Circuit Algorithm. Modify A by appending a row:

$$\widehat{A} = \begin{pmatrix} A \\ n_1^T \end{pmatrix}.$$

A complete matching in $\widehat{A}$ is obtained by augmenting a complete matching in A with the edge (n_1^T, u). Apply the circuit algorithm to find a null vector n_2 of $\widehat{A}$. Then n_2 is a null vector of A and is orthogonal to n_1. This procedure can be repeated until an orthogonal basis is obtained.

We believe it is unlikely that this algorithm will find sparse bases; to justify our pessimism, we need to characterize the relationship between columns of A in a circuit in terms of a block lower triangular decomposition of square matrices.

This decomposition was discovered by Dulmage and Mendelsohn (1963). It decomposes a square matrix M as

$$\begin{pmatrix} M_{11} & 0 & 0 & \dots & 0 \\ M_{21} & M_{22} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & & 0 \\ M_{k1} & M_{k2} & & \dots & M_{kk} \end{pmatrix},$$

where each diagonal submatrix M_{ii} is square, and has its column set C_i and its row set R_i perfectly matched to each other. Each set C_i will be called a Dulmage-Mendelsohn (D-M) block. The number of such blocks, k, and the partitions $\{C_1, \ldots, C_k\}$ and $\{R_1, \ldots, R_k\}$ are unique for a given matrix M. Duff (1977) has implemented this decomposition for efficient solution of sparse unsymmetric systems of equations.

This decomposition can be obtained by finding the strong components of a directed graph associated with M. It can also be found from a perfect matching $\mathcal{M}$ of M as described below. Define a *tour* as a sequence of edges in the bipartite graph of M, (c_0, r_0), (r_0, c_1), $\ldots$, (c_p, r_p), (r_p, c_0) that begins at ends at a column vertex c_0. Edges can be repeated in this sequence. A tour is $\mathcal{M}$*-alternating* if matched and unmatched edges alternate in the sequence. An equivalence relation is defined on the columns of M as follows: two columns are related if there is an $\mathcal{M}$ alternating tour between them. Then the column sets C_1, C_2, $\ldots$, C_k in the D-M decomposition are the equivalence classes of this relation. We will need this fact to prove the results of this section. A detailed discussion of this matching method for finding the D-M decomposition may be found in Pothen (1984).

We can characterize the columns in $n(u)$ in terms of the D-M blocks of the matched submatrix M. We prove that if one column of a D-M block C_i is in the circuit $n(u)$, then all of them are in it. Further, if a block C_i has nonzeros in a row set R_j with $j > i$, then all columns in the D-M block C_j are in a circuit containing the block C_i.

Theorem 5.1. *Let u be an unmatched column of A. If c and d are columns in C_i, and if $u \xrightarrow{M} c$, then $u \xrightarrow{M} d$.*

Proof: There is an $\mathcal{M}$-alternating tour from c to d by the definition of C_i. The $\mathcal{M}$-alternating path from u to c concatenated with the tour contains an $\mathcal{M}$-alternating path from u to d. ∎

Theorem 5.2. *Let u be an unmatched column of A. Let $u \xrightarrow{M} c$ for $c \in C_i$, and for some $j > i$, let $d \in C_j$. If M_{ji} contains at least one nonzero element, then $u \xrightarrow{M} d$.*

Proof: Since M_{ji} is nonzero, there is an edge between a column $e \in C_i$ and a row $r \in R_j$. Let r be matched in $\mathcal{M}$ to column f of C_j. From Theorem 5.1, $u \xrightarrow{M} e$. This path can be continued by the edges $(e, r), (r, f)$ to $f \in C_j$. Again by Theorem 5.1, since f and d are in C_j, $u \xrightarrow{M} d$. ∎

We now indicate how the D-M blocks of M change on augmentation with a null vector $n(u)$. A row n_1^T, which has nonzeros in those columns of A contained in $n(u)$, is added to A. A complete matching in $\widehat{A}$ can be found by augmenting $\mathcal{M}$ with the nonzero (n_1^T, u); thus the new row n_1^T is matched to u. We also update $\widehat{M} = M + u$, and $\widehat{U} = U - u$. The Circuit Algorithm can now applied to an unmatched column in $\widehat{U}$.

Theorem 5.3. *Let $K = \{1, \ldots, k\}$, and the D-M blocks of M be $C_1, \ldots, C_k$. For a null vector $n(u)$, let $I = \{i : C_i \subset n(u)\}$. Then $\widehat{M}$ has the D-M blocks $\{C_j : j \in K - I\}$, and $\cup_{i \in I} C_i$.*

Proof: Let the column sets $C_\ell \subset n(u)$ and $C_m \subset n(u)$. Then from Theorem 5.1, $u \xrightarrow{M} C_\ell$ and $u \xrightarrow{M} C_m$. The sequence of edges

$$u \xrightarrow{M} C_\ell,\ C_\ell \xrightarrow{M} n_1^T,\ (n_1^T, u),\ u \xrightarrow{M} C_m,\ C_m \xrightarrow{M} n_1^T,\ (n_1^T, u)$$

is an $\mathcal{M}$-alternating tour between columns in C_ℓ and C_m. Hence C_ℓ and C_m are in the same D-M block of $\widehat{M}$. ∎

This implies that at each stage of the algorithm the D-M blocks in a circuit merge to form one bigger block. Hence our pessimism about being able to compute sparse orthogonal null bases by this algorithm; however, we have no computational evidence to corroborate our pessimism.

6. Sparsest Orthogonal Null Bases

Sparsest null bases can be characterized by the greedy algorithm (Coleman and Pothen (1985)) that chooses, at each step, a sparsest null vector to augment a partial basis. But a greedy strategy will not find a sparsest orthogonal null basis. A counter-example is given in Pothen (1984). Unfortunately, for general sparse matrices, we do not know how to identify or construct sparsest orthogonal null bases.

In this section, we construct sparsest orthogonal bases for two special cases: an n-vector and a $t \times n$ dense matrix. Throughout this section, we assume that arithmetic on two nonzeros always yields a nonzero, unless numeric values are chosen to zero one of them. For a real number a, $\lceil a \rceil$ is the smallest integer greater than or equal to a, and $\lfloor a \rfloor$ is the greatest integer smaller than or equal to a.

An n-vector. Let $x = (x_1, \ldots, x_n)$ be a vector with n components, and consider computing the LQ decomposition of x. Let

$$xQ = (\times\ \underbrace{0 \ldots 0}_{n-1}), \tag{1}$$

where Q is an $n \times n$ orthogonal matrix formed as a product of *exactly* $n-1$ Givens rotations. The number of nonzeros in Q depends on the order in which the elements of the vector are 'zeroed'.

Consider the following divide and conquer strategy which splits an n-vector x into a k-vector $x^1 = (x_1 \ldots x_k)$ and an $(n-k)$-vector $x^2 = (x_{k+1} \ldots x_n)$, where $1 \le k \le n-1$. Suppose we have solved the two subproblems of constructing the sparsest orthogonal matrices, Q_1 and Q_2, where

$$\begin{aligned} x^1 Q_1 &= (\times 0 \ldots 0), \\ x^2 Q_2 &= (\times 0 \ldots 0). \end{aligned} \tag{2}$$

Then

$$(x_1 \quad x_2) \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} = \Big(\underbrace{\times 0 \ldots 0}_{k} \underbrace{\times 0 \ldots 0}_{n-k} \Big).$$

We now obtain the matrix Q from

$$Q = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} G_{1,k+1}. \tag{3}$$

The last Givens rotation uses x_1 to zero x_{k+1}.

We can apply this strategy recursively to the two subvectors x^1 and x^2. When a subvector has exactly two nonzeros, then we apply a Givens rotation to to zero one of the elements. Denote the first column of Q_1 by q_1.

Lemma 6.1. *All k components of q_1 are nonzero.*

Proof: Since

$$x^1 I = x^1, \qquad \text{and} \quad x^1 Q_1 = (\times 0 \ldots 0),$$

where I is the k-dimensional identity matrix, we can consider Q_1 as being formed from linear combinations of columns of the identity matrix. When x_i is used to zero x_j, the Givens rotation G_{ij} forms a linear combination of the i-th and j-th columns of the partially formed Q. When we complete the sequence of Givens rotations to obtain Q_1, only the first component of x^1 is nonzero; hence each of the last $k-1$ columns of the identity matrix has a nonzero coefficient in the linear combination that gives q_1. Since we have assumed that arithmetic on two nonzeros produces a nonzero, it follows that all components of q_1 are nonzero. ∎

From the lemma, computing a sparsest Q in (1) solves the problem of computing the sparsest orthogonal null basis of an n-vector.

We now show that any strategy that computes Q as a product of $n-1$ Givens rotations can be considered a divide and conquer strategy. Thus Theorem 6.2 will characterize a sparsest Q obtainable as a product of $n-1$ Givens rotations.

In any strategy, when G_{ij} is applied to the vector x to zero x_j using x_i, we hold x_i *responsible* for zeroing x_j. If at a later stage x_ℓ zeros x_i, we will hold x_ℓ *responsible* for zeroing both x_i and x_j. Consider the last Givens rotation in any strategy. At this stage, two nonzeros remain in the transformed vector x, and the rotation uses one to zero the other. Permute the nonzero that will survive this stage to the first column position of x. Also, permute all elements this nonzero is responsible for zeroing (prior to this stage) to the positions $x_2, \ldots, x_{k-1}$. Similarly, permute the nonzero that gets zeroed in the last stage to x_k, and all nonzeros it is responsible for zeroing to the positions $x_{k+1}, \ldots, x_n$. Clearly, the permutations do not change the number of nonzeros in Q. Thus, any strategy is a divide and conquer strategy.

Let $\log_2 n$ denote the logarithm of a number n to the base 2, and let $g(n) \equiv 2\left(n - 2^{\lfloor \log_2 n \rfloor}\right)$ represent a small correction that takes care of the aberration caused by the floor and ceiling functions.

Theorem 6.2. *The divide and conquer strategy with $k = \lceil n/2 \rceil$ leads to the sparsest possible orthogonal matrix Q in equation (1). This matrix has $n\lfloor \log_2 n \rfloor + n + g(n)$ nonzeros.*

Proof: Consider the last Givens rotation, $G_{1,k+1}$ in (3), in the divide and conquer strategy. By Lemma 6.1, the first column of Q_1 has k nonzeros, and that of Q_2 has $n-k$ nonzeros. The final Givens rotation forms a linear combination of the first and $(k+1)$-th columns of the partially formed orthogonal matrix. Thus, this rotation creates n additional nonzeros in the orthogonal matrix.

Let $f(n)$ represent the number of nonzeros in a sparsest Q in (1). If Q is to be sparsest possible, Q_1 and Q_2 have to be sparsest possible matrices that satisfy equation (2). Hence the matrix Q_1 has $f(k)$ and Q_2 has $f(n-k)$ nonzeros. This leads to the recurrence

$$f(n) = \min_{1 \le k \le n-1} f(k) + f(n-k) + n, \tag{4}$$

with initial conditions, $f(1) = 1$, and $f(2) = 4$. This recurrence is solved using a technique in Greene and Knuth (1982). The details may be found in Pothen (1984). The solution is

$$k = \lceil n/2 \rceil,$$

$$f(n) = n\lfloor \log_2 n \rfloor + n + g(n).$$

∎

The densest possible Q in equation (1) is an upper Hessenberg matrix. A proof of the following theorem can be found in Pothen (1984).

Theorem 6.3. *The densest Q that satisfies equation (1) has $(n^2+3n-2)/2$ nonzeros.*

Rectangular matrices. We can generalize Theorems 6.2 to matrices with t rows and n columns, where $t < n$. As for n-vectors, the problem of computing sparsest orthogonal null bases can be reduced to the problem of computing the sparsest Q in the LQ factorization of A.

For ease of analysis, let $n = it$ for some integer $i \geq 2$. (When n is not a multiple of t, a more involved analysis leads to a result with the same leading term as the one below.) Partition A as $A = (A_1\, A_2)$, where A_1 has kt columns, and A_2 has $(i-k)t$ columns, for an integer $1 \leq k \leq i-1$. Suppose we have constructed LQ decompositions for A_1 and A_2,

$$A_1 Q_1 = (L_1 \quad 0),$$
$$A_2 Q_2 = (L_2 \quad 0).$$

We partition the orthogonal matrix Q_i as $Q_i = (Y_i\, Z_i)$, where Y_i is the submatrix formed by the first t columns of Q_i.

The submatrices Z_1 and Z_2 form $(i-2)\,t$ null vectors of A. The last t null vectors in a null basis of A can be obtained from the equation

$$(A_1 \quad A_2) \begin{pmatrix} Y_1 & 0 \\ 0 & Y_2 \end{pmatrix} = (L_1 \quad L_2).$$

Postmultiply this equation with Givens rotations so as to get

$$(A_1 \quad A_2)\, Q_{12} = (L_{12} \quad 0). \tag{5}$$

The last t columns of Q_{12} are the desired null vectors of A. This strategy can be recursively applied to A_1 and A_2.

It is easily seen that the last such stage in equation (5) adds $nt = it^2$ nonzeros to the basis. Let $f(i)$ represent the number of nonzeros in a sparsest Q when A has it columns. The recurrence analogous to equation (4) is

$$f(i) = \min_{1 \leq k \leq i-1} f(k) + f(i-k) + it^2,$$

with initial conditions $f(1) = t^2$, and $f(2) = 4t^2$. If we divide the above equation by t^2, it transforms to equation (4), and hence the best splitting strategy is given by $k = \lceil i/2 \rceil$, and

$$\begin{aligned} f(i) &= t^2\ (i\lfloor \log_2 i \rfloor + i + g(i)) \\ &= nt\lfloor \log_2 n/t \rfloor + nt + t^2 g(n/t). \end{aligned}$$

This strategy for computing the LQ factorization can be efficiently implemented on a computer with parallel architecture. The matrix is partitioned into $t \times t$ submatrices, with each submatrix assigned to a single processor. In the first step, each processor computes the factorization of its submatrix. In the second step, processors pair up to compute the LQ factorization of a $t \times 2t$ submatrix, as in (5). At each succeeding step, by recursive doubling, the size of the factored submatrix doubles, until in $\lceil \log_2 n/t \rceil$ steps the factorization is computed for the entire matrix.

The generalization of Theorem 6.3 to $t \times n$ matrices shows that the a densest Q has $(n^2 + 3nt - 2t^2)/2$ nonzeros.

7. Conclusions

We have shown that matchings can be used to identify dependent sets of columns in a matrix, and thereby nonzeros in a null vector. These dependent sets are formed by choosing a start column and adding columns to it, one by one. Matchings help us in making good choices for columns to add so that a sparse null vector is obtained. This is accomplished by weighting columns and choosing a column of minimum weight to match to a row.

The resulting algorithms to compute null vectors have two phases: a combinatorial phase, in which dependent sets are identified, and a numeric phase, in which the coefficients of the null vector are computed. The time required for the second phase clearly dominates that of the first.

We have also focused attention on the structures of the null bases we construct. To ensure linear independence of the computed null vectors, the null bases are restricted to be triangular or fundamental.

To compute a fundamental basis, we need to ensure that the matched submatrix has full row rank. Only one sparse LU factorization of the matched submatrix and $(n - t)$ backsolves are needed to compute the basis. However, since all the null vectors are computed from a fixed initial matching, it is difficult to assign weights 'globally' to columns in an intelligent way.

For each null vector in a triangular basis, we need to find a perfect matching in a submatrix, compute its LU factors, and perform a backsolve. In this case, a more intelligent dynamic column weighting strategy (of the Modified Circuit Algorithm) is possible to ensure sparsity in each null vector. Triangular bases we have computed are consistently sparser than fundamental bases; but the latter can be computed about two to ~~four~~ five times faster.

The conditioning of the null bases seems to depend on the conditions of the constraint matrices. When the latter were well-conditioned, the null bases were well-conditioned also, and there was less than a ten fold increase in estimated condition numbers. For one of the badly conditioned problems, the condition number increased a hundred fold. Developing algorithms that can control the conditioning of the null bases is an important open problem.

We have investigated the construction of sparse orthogonal null bases. A matching based algorithm is presented, but it seems unlikely that the computed bases will be sparse. We have shown how sparsest orthogonal bases can be computed for an n-vector and a $t \times n$ dense matrix by a divide and conquer strategy for LQ factorizations.

Throughout this paper, we have assumed A has full row rank. Algorithm 2 can be modified to work in the rank deficient situation also. This can be done by rejecting unmatched rows in a maximum matching, since these are structurally dependent rows. Algorithm 2 can then proceed until all unmatched columns have been used to compute null vectors. Then an LU factorization of M, the completely matched submatrix, can be used to identify the rest of the dependent columns.

Both Algorithms 1 and 2 compute null vectors in a dependent set by computing LU factorizations; the LUSOL routines that do this have to decide when a column should be declared dependent in the factorization. This is a difficult numerical problem. Thus computing null bases is not immune from the difficulties associated with numerical rank determination.

Gilbert and Heath (1986) have implemented a matching based algorithm (GHM) and the turnback algorithm (GHT) to compute null bases. Matching based algorithms outperform turnback on two counts: They compute sparser null bases since columns in a dependent set are chosen by a more sophisticated combinatorial criterion, rather than by a column numbering. They also compute null bases faster, since dependent sets are identified combinatorially, rather than by a QR factorization. GHM was on the average about five times faster than GHT. On larger problems, it is expected that this speedup should be even greater.

The differences between Algorithm 2 and GHM are two-fold. We choose start columns by matchings, they, by an initial QR factorization of A. More important, there is a clear separation between the combinatorial and numeric phases in our algorithm; they maintain a QR factorization of the active submatrix as each column is added, and choose columns to add by matching. This helps them identify numerical dependences not predictable structurally. But the price they pay is the storage of the dense factors of the active submatrix. We store the active submatrix, which is sparse, and factor only the dependent set by a sparse LU factorization routine. Thus no dense matrix storage is required by Algorithm 2.

From our results on test problems from Gilbert and Heath (1986), the intermediate storage required by Algorithm 2 is a fraction of the storage needed for A. In both GHM and GHT, the intermediate dense matrix storage needed is substantially greater than the storage required for A. This should cause a serious limitation on the size of null basis problems that can be solved by GHM and GHT.

On the structural problems, the running times of Algorithm 2 and GHM are quite close. GHM has a slight edge over Algorithm 2 in sparsity. But on the larger linear program in this set, Algorithm 2 runs more than twice as fast as GHM, and about twenty times faster than GHT. On larger linear and nonlinear programs, we expect this edge to be greater. However, since we have not implemented their algorithms, we have no comparisons of performance on the linear programming problems in Table 2.

We expect our algorithms to compute quite sparse bases when the matrix elements in A are 'reasonably' random, and the matrix has no additional 'structure'. In particular, when A is the vertex-edge incidence matrix of a directed graph, a circuit corresponds to a cycle in the graph, and a basis for the cycle space forms a null basis of A. A cycle has an equal number of nonzero columns and rows of the vertex-edge incidence matrix, unlike the circuits in this paper. Our model of a circuit is a submatrix with a complete matching which has one more nonzero column than its number of nonzero rows. Sparser cycle bases could probably be obtained by a model that exploits this additional 'structure'.

Equilibrium matrices from structural optimization have, like vertex-edge matrices of graphs, additional structure. For these problems, most circuits have equal numbers of rows and columns. By considering 'equilibrium graphs', bipartite graphs associated with equilibrium matrices, it is possible to exploit the structure in these problems, and to model circuits more accurately (Pothen(1986)). This approach yields a new algorithm to compute null bases for equilibrium matrices. In some cases even sparsest null bases can be characterized.

Computing sparse cycle bases is important in solving nonlinear programs with network constraints (Dembo (1983)). Little is known about computing sparsest cycle bases. Deo, Prabhu, and Krishnamoorthy (1982) show that it is NP-complete to find sparsest fundamental cycle bases, and have proposed heuristic algorithms (and Pascal programs) to find sparse fundamental bases.

Acknowledgements

Our thanks to John Gilbert and Mike Heath, for sharing their experiences in designing their null basis code, which helped us with our implementation; to Mike Saunders, for sending the LUSOL package and the lp test problems to us, and for being available with advice; and to the referees, for their valuable suggestions on our original paper.

Hall property (HP)	every subset of k rows has nonzeros in at least k columns
Strong Hall property (SHP)	every subset of k rows has nonzeros in at least $k+1$ columns
weak Haar property (wHP)	every subset of columns C has $m(C) = \text{rank}(C)$

Table 1.
Summary of properties.

Problem	*rows*	*cols*	*nonzeros*	*density (%)*
murty	12	30	56	15.6
afiro	27	51	102	7.4
adlittle	56	138	424	5.5
share2b	96	162	777	5.0
share1b	117	253	1179	4.0
beaconfd	173	295	3408	6.7
israel	174	316	2443	4.4
brandy	193	303	2202	3.8
e226	223	472	2768	2.6
capri	271	482	1896	1.5
bandm	305	472	2494	1.7
stair	356	614	4013	1.8
etamacro	400	816	2537	0.8

Table 2.

Test Problems.

Problem	Null Basis rows	cols	nonzeros	density (%)	Time (seconds) total	match	factor	solve
murty	30	18	62	11.5	0.6	.02	.07	0.5
afiro	51	24	112	9.2	0.9	.05	.08	0.8
adlittle	138	82	500	4.4	3.5	.07	0.2	3.2
share2b	162	66	736	6.9	4.6	0.1	1.0	4.0
share1b	253	136	2264	6.6	13.7	0.7	1.7	11.3
beaconfd	295	122	1789	5.0	13.9	0.7	1.1	11.9
israel	316	142	2411	5.4	12.1	.02	0.4	11.6
brandy	303	110	4758	14.3	22.8	1.8	2.3	18.6
e226	472	249	3449	2.9	20.3	0.3	0.6	19.2
capri	482	211	3478	3.4	27.6	4.5	2.5	20.3
bandm	472	167	2306	2.9	19.5	0.7	0.1	17.4
stair	614	258	5378	3.4	35.7	1.2	1.2	33.2
etamacro	816	416	3929	1.2	39.1	2.8	2.5	33.8

Table 3.

Fundamental Null Bases.

Problem	Null Basis rows	cols	nonzeros	density (%)	Time (seconds) total
murty	30	18	62	11.5	0.4
afiro	51	24	108	8.8	0.8
adlittle	138	82	486	4.3	3.6
share2b	162	66	686	6.4	5.0
share1b	253	136	1425	4.1	19.8
beaconfd	295	122	1581	4.4	70.4
israel	316	142	2118	4.7	34.4
brandy	303	110	2535	7.6	91.8
e226	472	249	2742	2.3	57.4
capri	482	211	2850	2.8	50.3
bandm	472	167	1941	2.5	26.4
stair	614	258	5094	3.2	59.9
etamacro	816	416	3563	1.0	59.5

Table 4.

Triangular Null Bases.

Problem	*rows*	*cols*	*nonzeros*	*density (%)*
frame2d	27	45	93	7.7
lp1	57	97	465	8.4
frame3d	72	144	304	2.9
wheel	96	120	420	3.6
wrench	112	216	490	2.0
lp2	118	225	1182	4.5
mixed1	171	320	906	1.7

Table 5.

Test problems from Gilbert and Heath (1986).

	Gilbert and Heath				Algorithm 2		
Problem	*algorithm*	*nonzeros*	*time*	*storage*	*nonzeros*	*time*	*storage*
frame2d	matching	76	0.63	72	79	0.63	16
	turnback	76	2.30	288			
lp1	matching	367	10.50	1680	351	15.3	136
	turnback	391	28.02	2550			
frame3d	matching	317	2.95	168	310	3.6	35
	turnback	452	66.12	2064			
wheel	matching	488	10.45	756	518	5.5	103
	turnback	503	17.32	1560			
wrench	matching	518	26.98	3782	588	31.8	139
	turnback	544	58.38	5112			
lp2	matching	1363	103.68	8160	1379	39.1	680
	turnback	1531	773.15	13570			
mixed1	matching	1101	60.80	3540	1210	78.9	169
	turnback	1518	288.87	10506			

Table 6.

Results on problems from Gilbert and Heath (1986).

The storage reported is the intermediate storage required to obtain a dependent set. For the Gilbert and Heath algorithms, this is the maximum size of the dense matrix storage needed for the QR factors of the active submatrix. For Algorithm 2, this is the maximum number of nonzeros in the sparse LU factorization of the dependent set.

References

M.W. Berry, M.T. Heath, I. Kaneko, M. Lawo, R.J. Plemmons, and R.C. Ward, *An algorithm to compute a sparse basis of the null space*, Numer. Math., 47 (1985), pp. 483-504.

M.W. Berry and R.J. Plemmons, *Computing a banded basis of the null space on the Denelcor HEP multiprocessor*, Proc. AMS/SIAM Summer Conference on Linear Algebra in Systems Theory, AMS Series on Contemp. Math. 1985.

Thomas F. Coleman, *On chordal preconditioners for large scale optimization*, Technical Report (to appear), Computer Science Dept., Cornell Univ., 1986.

Thomas F. Coleman, Anders Edenbrandt, and John R. Gilbert, *Predicting fill for sparse orthogonal factorization*, J. ACM, (to appear) 1985.

Thomas F. Coleman and Jorge J. Moré, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM J. Num. Anal., 20 (1983), pp. 187–209.

Thomas F. Coleman and Jorge J. Moré, *Estimation of sparse Hessian matrices and graph coloring problems*, Math. Prog., 28 (1984), pp. 243–270.

Thomas F. Coleman and Alex Pothen, *The Null Space Problem I: Complexity*, SIAM J. Alg. Disc. Meth., (to appear) 1985.

Ron S. Dembo, *A primal truncated Newton algorithm with application to large-scale network optimization*, Working Paper B-72, Yale School of Orgn. and Man., 1983.

Narsingh Deo, G.M. Prabhu, and M.S. Krishnamoorthy, *Algorithms for generating fundamental cycles in a graph*, ACM Trans. Math. Software, 8 (1982), pp. 26-42.

I.S. Duff, *MA28–A set of Fortran subroutines for sparse unsymmetric linear equations*, AERE Report R8730, Harwell, England, 1977.

I.S. Duff, *On algorithms for obtaining a maximum transversal*, ACM Trans. Math. Software, 7 (1981), pp. 315–330.

Alan George and Joseph W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, Englewood Cliffs, 1981.

John R. Gilbert and Michael T. Heath, *Computing a sparse basis for the null space*, Technical Report 86-730, Computer Science Dept., Cornell Univ., 1986.

P.E. Gill, W. Murray, and M.H. Wright, *Practical Optimization*, Academic Press, New York, 1981.

P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright, *Maintaining LU factors of a general sparse matrix*, Technical Report (to appear), Systems Optimization Lab., Stanford Univ., 1986.

D. Goldfarb and S. Mehrotra, *A relaxed version of Karmarkar's method*, Technical Report, Dept. I.E.& O.R., Columbia Univ., 1985.

Daniel H. Greene and Donald E. Knuth, *Mathematics for the Analysis of Algorithms*, Birkhäuser, Boston, 1982.

J.E. Hopcroft and R.M. Karp, *A $n^{2.5}$ algorithm for maximum matchings in bipartite graphs*, SIAM J. Comput., 2 (1973), pp. 225-231.

I. Kaneko, M. Lawo and G. Thierauf, *On computational procedures for the force method*, Inter. J. Numer. Meth. in Eng., 18 (1982), pp. 1469-1495.

Eugene L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New York, 1976.

Bruce A. Murtagh and Michael A. Saunders, *MINOS 5.0 user's guide*, Technical Report SOL 83-20, Systems Optimization Lab., Stanford Univ., 1983.

Katta G. Murty, *Linear Programming*, John Wiley and Sons, New York, 1983.

Christos H. Papadimitriou and Kenneth Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, N.J. 1982.

Alex Pothen, *Sparse null bases and marriage theorems*, Ph.D. Thesis, Cornell Univ., Ithaca, New York, 1984.

Alex Pothen, *Equilibrium graphs in structural optimization*, Technical Report (to appear), Computer Science Dept., Penn. State Univ., 1986.

J.K. Reid, *Fortran subroutines for handling sparse linear programming bases*, AERE Report R8269, Harwell, England, 1976.

J.K. Reid, *A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases*, Math. Prog., 24 (1982), pp. 55–69.

D.F. Shanno and R.E. Marsten, *On implementing Karmarkar's method*, Working Paper 85-01, Dept. Man. Info. Systems, Univ. Arizona, 1985.

M.N. Thapa, *Optimization of unconstrained functions with sparse Hessian matrices*, Math. Prog., 29 (1984), pp. 156–186.

A. Topcu, *A contribution to the systematic analysis of finite element structures through the force method*, Ph.D. Thesis, Univ. of Essen, Essen, Germany, 1979. (in German.)

P. Wolfe, *The reduced gradient method*, The RAND Corporation, 1962, unpublished manuscript.