A Column Approximate Minimum Degree Ordering Algorithm

TIMOTHY A. DAVIS
University of Florida
JOHN R. GILBERT
University of California, Santa Barbara
STEFAN I. LARIMORE
Microsoft, Inc.
and
ESMOND G. NG
Lawrence Berkeley National Laboratory

Sparse Gaussian elimination with partial pivoting computes the factorization $\mathbf{PAQ} = \mathbf{LU}$ of a sparse matrix $\mathbf{A}$, where the row ordering $\mathbf{P}$ is selected during factorization using standard partial pivoting with row interchanges. The goal is to select a column preordering, $\mathbf{Q}$, based solely on the nonzero pattern of $\mathbf{A}$, that limits the worst-case number of nonzeros in the factorization. The fill-in also depends on $\mathbf{P}$, but $\mathbf{Q}$ is selected to reduce an upper bound on the fill-in for any subsequent choice of $\mathbf{P}$. The choice of $\mathbf{Q}$ can have a dramatic impact on the number of nonzeros in $\mathbf{L}$ and $\mathbf{U}$. One scheme for determining a good column ordering for $\mathbf{A}$ is to compute a symmetric ordering that reduces fill-in in the Cholesky factorization of $\mathbf{A}^\mathsf{T}\mathbf{A}$. A conventional minimum degree ordering algorithm would require the sparsity structure of $\mathbf{A}^\mathsf{T}\mathbf{A}$ to be computed, which can be expensive both in terms of space and time since $\mathbf{A}^\mathsf{T}\mathbf{A}$ may be much denser than $\mathbf{A}$. An alternative is to compute $\mathbf{Q}$ directly from the sparsity structure of $\mathbf{A}$; this strategy is used by MATLAB's COLMMD preordering algorithm. A new ordering algorithm, COLAMD, is presented. It is based on the same strategy but uses a better ordering heuristic. COLAMD is faster and computes better orderings, with fewer nonzeros in the factors of the matrix.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra – *linear systems (direct methods), sparse and very large systems*; G.4 [**Mathematics of Computing**]: Mathematical Software – *algorithm analysis, efficiency*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: sparse nonsymmetric matrices, linear equations, ordering methods

Author's address: Timothy A. Davis, Computer and Info. Sci. and Eng. Dept., University of Florida, Gainesville, FL, USA. http://www.cise.ufl.edu/∼davis.
This work was supported in part by the National Science Foundation under grant number DMS-9803599; in part by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098; and in part by DARPA under contract DABT63-95-C-0087.

1. INTRODUCTION

Sparse Gaussian elimination with partial pivoting computes the factorization $\mathbf{PAQ} = \mathbf{LU}$ for the sparse nonsymmetric matrix $\mathbf{A}$, where $\mathbf{P}$ and $\mathbf{Q}$ are permutation matrices, $\mathbf{L}$ is a lower triangular matrix, and $\mathbf{U}$ is an upper triangular matrix. Gilbert and Peierls [1988] have shown that sparse partial pivoting can be implemented in time proportional to the number of floating-point operations required. The method is used by the sparse matrix package, SuperLU [Demmel et al. 1999], and was the first method for sparse LU factorization in MATLAB [Gilbert et al. 1992]. The solution process starts by finding a sparsity-preserving permutation $\mathbf{Q}$. Next, the permutation $\mathbf{P}$ is selected during numerical factorization using standard partial pivoting with row interchanges. The permutation $\mathbf{P}$ is selected without regard to sparsity. Our goal is to compute a sparsity-preserving permutation $\mathbf{Q}$ solely from the pattern of $\mathbf{A}$ that limits the worst-case number of nonzeros in the LU factorization $\mathbf{PAQ} = \mathbf{LU}$. The fill-in also depends on $\mathbf{P}$, but $\mathbf{Q}$ is selected to reduce an upper bound on the fill-in for any subsequent choice of $\mathbf{P}$. Our resulting code has been incorporated in SuperLU, UMFPACK, MATLAB Version 6.0 (as the `colamd` function), and MATLAB Version 6.5 (as the pre-ordering for the backslash operator, `x = A\b`).

Section 2 provides the theoretical background for our algorithm. In Section 3, we describe symbolic LU factorization (with no column interchanges) and our upper bound on the nonzero pattern of the submatrices that arise during an outer-product LU factorization, as a precursor to the COLAMD ordering algorithm presented in Section 4. Section 4 also describes the various metrics for selecting columns that we evaluated in the design of the code. Experimental results for square nonsymmetric matrices, rectangular matrices, and symmetric matrices are presented in Section 5. Section 6 presents our conclusions and describes the availability of our software.

Our notation is as follows. Set subtraction is denoted by the "\" operator. We use $|...|$ to denote either the absolute value of a scalar, the number of nonzero entries in a matrix, or the size of a set. The usage will be clear in context. The structure of a matrix $\mathbf{A}$, denoted by $\mathtt{Struct}(\mathbf{A})$, is a set that contains the locations of the nonzero entries in $\mathbf{A}$; that is, $\mathtt{Struct}(\mathbf{A}) = \{(i,j) : \mathbf{A}_{ij} \neq 0\}$. For a vector, $\mathtt{Struct}(\mathbf{x}) = \{i : \mathbf{x}_i \neq 0\}$. Throughout this paper, and in the algorithms, we assume that exact numerical cancellations do not occur. Some entries in the matrix may happen to become zero during factorization (because of accidental cancellation); we still refer to these as "nonzero" entries.

2. $\mathbf{A}^\mathsf{T}\mathbf{A}$ ORDERINGS

Let $\mathbf{A}$ be a given nonsymmetric matrix and assume that it is nonsingular. It follows from Duff [1981] that the rows (or the columns) of $\mathbf{A}$ can be permuted so that the diagonal entries of the permuted $\mathbf{A}$ are all nonzero. We therefore assume throughout this paper that the given matrix $\mathbf{A}$ has been permuted accordingly so that it has a zero-free diagonal.

Suppose that $\mathbf{L}$ and $\mathbf{U}$ are the triangular factors obtained when Gaussian elimination with partial pivoting is applied to $\mathbf{A}$. That is, $\mathbf{PA} = \mathbf{LU}$, for some row permutation $\mathbf{P}$. Consider the symmetric positive definite matrix $\mathbf{A}^\mathsf{T}\mathbf{A}$, which has a Cholesky factorization $\mathbf{A}^\mathsf{T}\mathbf{A} = \mathbf{L}_C\mathbf{L}_C^\mathsf{T}$ with $\mathbf{L}_C$ being lower triangular. George and

Ng [1985] showed that if $\mathbf{A}$ has a zero-free diagonal, then the pattern of $\mathbf{L}_C + \mathbf{L}_C^\mathsf{T}$ includes the patterns of $\mathbf{L}$ and $\mathbf{U}$, regardless of the row permutation used in partial pivoting. This is because the structure of row i of $\mathbf{A}^\mathsf{T}\mathbf{A}$ is the union of the structure of all rows having a nonzero entry in column i of $\mathbf{A}$. We summarize the result in the following theorem.

THEOREM 2.1 [GEORGE AND NG 1985]. *Let $\mathbf{A}$ be a nonsingular and nonsymmetric matrix that has a zero-free diagonal. Let $\mathbf{L}_C$ denote the Cholesky factor of $\mathbf{A}^\mathsf{T}\mathbf{A}$. Let $\mathbf{L}$ and $\mathbf{U}$ be the LU factors of $\mathbf{A}$ obtained by partial pivoting. Then $\mathtt{Struct}(\mathbf{U}) \subseteq \mathtt{Struct}(\mathbf{L}_C^\mathsf{T})$, and the entries within each column of $\mathbf{L}$ can be rearranged to yield a triangular matrix $\hat{\mathbf{L}}$ with $\mathtt{Struct}(\hat{\mathbf{L}}) \subseteq \mathtt{Struct}(\mathbf{L}_C)$.*

Gilbert and Ng [1993] also showed that when $\mathbf{A}$ is a strong Hall matrix, the bound on $\mathbf{U}$ is tight in the following sense. [1]

THEOREM 2.2 [GILBERT AND NG 1993]. *Let $\mathbf{A}$ be a nonsingular and nonsymmetric matrix that has a zero-free diagonal. Assume that $\mathbf{A}$ is strong Hall. Let $\mathbf{L}_C$ denote the Cholesky factor of $\mathbf{A}^\mathsf{T}\mathbf{A}$. Let $\mathbf{L}$ and $\mathbf{U}$ be the LU factors of $\mathbf{A}$ obtained by partial pivoting. For any choice of $(i, j) \in \mathtt{Struct}(\mathbf{L}_C^\mathsf{T})$, there exists an assignment of numerical values to the nonzero entries of $\mathbf{A}$ such that $\mathbf{U}_{ij} \neq 0$.*

We use a tighter bound on the nonzero pattern of $\mathbf{L}$ and $\mathbf{U}$, which is described in detail in Section 3.

It is well known that the sparsity of $\mathbf{L}_C$ depends drastically on the way in which the rows and columns of $\mathbf{A}^\mathsf{T}\mathbf{A}$ are permuted [George and Liu 1981]. Thus, Theorems 2.1 and 2.2 suggest a way to reduce the amount of fill in $\mathbf{L}$ and $\mathbf{U}$ [George and Ng 1985; 1987]. Given a matrix $\mathbf{A}$, we first form the pattern of $\mathbf{A}^\mathsf{T}\mathbf{A}$. Then we compute a symmetric ordering $\mathbf{Q}$ of $\mathbf{A}^\mathsf{T}\mathbf{A}$ to reduce the amount of fill in $\mathbf{L}_C$. Finally, we apply the permutation $\mathbf{Q}$ to the columns of $\mathbf{A}$.

The main disadvantage with the approach above is the cost of forming $\mathbf{A}^\mathsf{T}\mathbf{A}$. Even if $\mathbf{A}$ is sparse, the product $\mathbf{A}^\mathsf{T}\mathbf{A}$ may contain more nonzeros than the LU factorization of $\mathbf{A}$. Consequently, the time and storage required to form the product may be high. The primary goal of this paper is to describe ordering algorithms that compute $\mathbf{Q}$ from the pattern of $\mathbf{A}$ without explicitly forming $\mathbf{A}^\mathsf{T}\mathbf{A}$.

3. SYMBOLIC LU FACTORIZATION

Our column ordering technique is based on a symbolic analysis of the conventional outer-product formulation of Gaussian elimination of the n-by-n matrix $\mathbf{A}$ with row interchanges to maintain numerical stability. Let $\mathbf{A}^{(0)} = \mathbf{A}$. For $k = 1, 2, \ldots, n-1$, let $\mathbf{A}^{(k)}$ denote the bottom right $(n-k)$-by-$(n-k)$ submatrix of $\mathbf{A}^{(k-1)}$ after the kth step of Gaussian elimination is performed. We assume that the rows and columns of $\mathbf{A}^{(k)}$ are labeled from $k+1$ to n.

At the k-th step of Gaussian elimination, one finds the entry with the largest magnitude in column k of $\mathbf{A}^{(k-1)}$ and swaps rows to place this entry on the diagonal. Column k of $\mathbf{L}$ is then a scaled version of column k of $\mathbf{A}^{(k-1)}$, and row k of $\mathbf{U}$ is row k of $\mathbf{A}^{(k-1)}$. The outer product of the column k of $\mathbf{L}$ and the row k of $\mathbf{U}$ is

[1] A matrix $\mathbf{A}$ is strong Hall if every set of k columns of $\mathbf{A}$, $1 \leq k \leq n-1$, contain at least $k+1$ nonzero rows. A strong Hall matrix is irreducible. See Coleman et al. [1986] for details.

subtracted from $\mathbf{A}^{(k-1)}$ to give $\mathbf{A}^{(k)}$. The result is the factorization $\mathbf{PA} = \mathbf{LU}$, where $\mathbf{P}$ is the permutation matrix determined by the row interchanges. When $\mathbf{A}$ is sparse, the update using the outer product may turn some of the zero entries into nonzero. These new nonzero entries are referred to as *fill-in*. The amount of fill-in that occurs depends on the order in which the columns of $\mathbf{A}$ are eliminated [George and Liu 1981]. Our goal, therefore, is to find a column ordering $\mathbf{Q}$ of the matrix $\mathbf{A}$ prior to numerical factorization that limits the worst-case number of nonzeros in the LU factorization $\mathbf{PAQ} = \mathbf{LU}$. The fill-in also depends on $\mathbf{P}$, but $\mathbf{Q}$ is selected to reduce an upper bound on the fill-in, for any subsequent choice of $\mathbf{P}$. Another application of column orderings is sparse QR factorization [Heggernes and Matstoms 1994].

In order to control the fill-in that occurs when $\mathbf{A}$ is sparse, our method needs to know the upper bounds of nonzero patterns of the matrices $\mathbf{A}^{(k)}$, for each k from 0 to $n-1$. The upper bound nonzero pattern of $\mathbf{A}^{(k)}$ is described below. From this symbolic LU factorization process, we can determine a column ordering $\mathbf{Q}$ that attempts to keep the factorization of $\mathbf{AQ}$ sparse as the factorization progresses.

The symbolic LU factorization process is best explained using the *row-merge tree* [Liu 1991], which describes the relationships between nonzero patterns of rows in the original matrix and the rows of the matrices $\mathbf{A}^{(k)}$. When the matrix has a zero-free diagonal and is strong-Hall, the row-merge tree is a slight modification of the elimination tree of the matrix $\mathbf{A}^{\mathsf{T}}\mathbf{A}$ (also referred to as the *column elimination tree*). For clarity, we will assume in this discussion that no column permutations are being made during the elimination phase. The symbolic factorization will account for all possible row permutations due to partial pivoting, however.

Let $\mathcal{A}_i$ denote the set of column indices of nonzero entries in row i of the original matrix $\mathbf{A}$. That is, $\mathcal{A}_i = \mathtt{Struct}(\mathbf{A}_{i*})$. Let $\mathcal{R}_k$ denote the upper bound nonzero pattern of the kth pivot row (excluding column index k itself). That is, $j \in \mathcal{R}_k$ means that the off-diagonal entry u_{kj} might become nonzero during the numerical LU factorization when the row ordering is determined.

At the first step of symbolic factorization, any original row i with a nonzero in column 1 might become the first pivot row. The set $\mathcal{R}_1$ is the union of these original rows, since we do not know which of these rows will be selected during numerical factorization. One row will become the pivot row and be removed from $\mathbf{A}^{(1)}$, and the rest remain in the matrix $\mathbf{A}^{(1)}$. The pivot column 1 is also removed from $\mathbf{A}^{(1)}$. Thus, we have

$$\mathcal{R}_1 = \left(\bigcup_{1 \in \mathcal{A}_i} \mathcal{A}_i \right) \setminus \{1\}.$$

Consider the candidate pivot rows $\{i : 1 \in \mathcal{A}_i\}$ that do not become pivotal. Fill-in from the outer-product update causes these rows to take on an upper bound nonzero pattern that is exactly equal to the pivot row pattern $\mathcal{R}_1$. Any column index j in the pattern of these rows will appear in $\mathcal{R}_1$ because of how $\mathcal{R}_1$ is constructed, and any column index j in $\mathcal{R}_1$ causes fill-in in each of these candidate rows. We can discard the sets $\mathcal{A}_i$ for all i such that $1 \in \mathcal{A}_i$, and represent the upper bound nonzero pattern of all of these rows with $\mathcal{R}_1$ instead. Since an original set $\mathcal{A}_i$ is

discarded as soon as it becomes a candidate pivot row, we can also write

$$\mathcal{R}_1 = \left(\bigcup_{1=\min \mathcal{A}_i} \mathcal{A}_i \right) \setminus \{1\}.$$

An example matrix $\mathbf{A}$ is shown in upper left corner of Figure 1. The first pivot row will be row 1 or 2, and $\mathcal{R}_1 = \{2, 4\}$. One of these two rows will be selected as a pivot, and the other will have a nonzero pattern of $\mathcal{R}_1 = \{2, 4\}$ after the first step of Gaussian elimination. The upper bound nonzero pattern of the matrix $\mathbf{A}^{(1)}$ is also shown in the figure, assuming that row 1 is selected as the first pivot row. For clarity in this example, we will assume that the diagonal is selected as pivot, but we will still use the upper bound pivot row patterns. The rest of Figure 1 illustrates the symbolic factorization process, and the row-merge tree.

Consider the second step. In general, entry 2 may or may not be in $\mathcal{R}_1$, but it is present in the example in Figure 1. This means that all the rows represented by $\mathcal{R}_1$ are also candidate pivot rows at step 2. The other candidate pivot rows at step 2 are all those rows that are zero in column 1 (otherwise they would be represented by $\mathcal{R}_1$) but nonzero in column 2. In our example, these are rows 4 and 5. We can form the union of all these candidate rows to find $\mathcal{R}_2$,

$$\mathcal{R}_2 = \mathcal{R}_1 \cup \left(\bigcup_{2=\min \mathcal{A}_i} \mathcal{A}_i \right) \setminus \{2\} = \{4, 5, 6, 8\}.$$

Because of fill-in, all candidate pivot rows take on the same upper bound on their nonzero pattern, $\mathcal{R}_2$. We can discard $\mathcal{R}_1$, since it is no longer needed to represent the nonzero patterns of the candidate rows at step 1. We can discard all the sets $\mathcal{A}_i$ for which $2 = \min \mathcal{A}_i$, since these rows also now have an upper bound equal to $\mathcal{R}_2$. The set $\mathcal{R}_2$ represents all the candidate rows at both step 1 and step 2, except for two of them that are selected as the first and second pivot rows. Because sets are discarded as soon as they are used, we can write

$$\mathcal{R}_2 = \left(\bigcup_{2=\min \mathcal{R}_i} \mathcal{R}_i \right) \cup \left(\bigcup_{2=\min \mathcal{A}_i} \mathcal{A}_i \right) \setminus \{2\}.$$

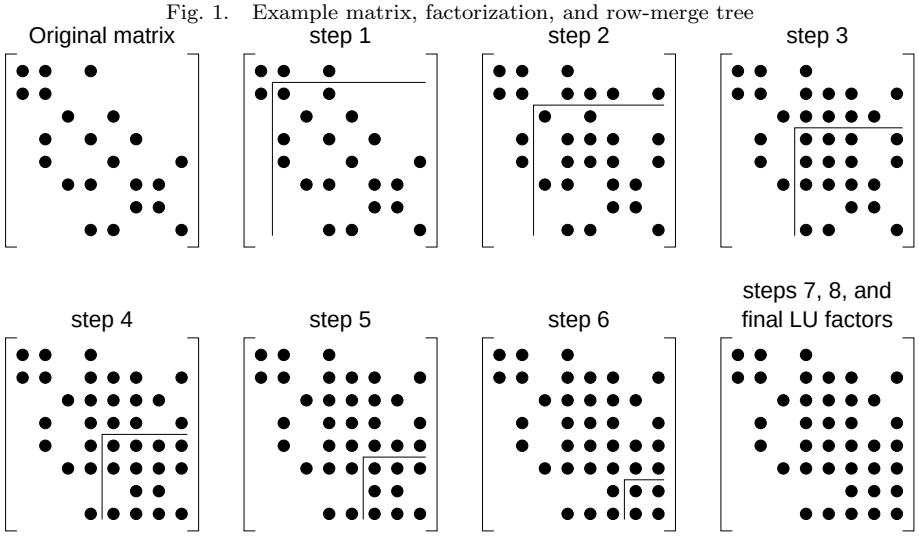
In this example, $\mathcal{R}_2$ represents the pattern of rows 4 and 5, since we assume rows 1 and 2 have been selected as pivot rows.

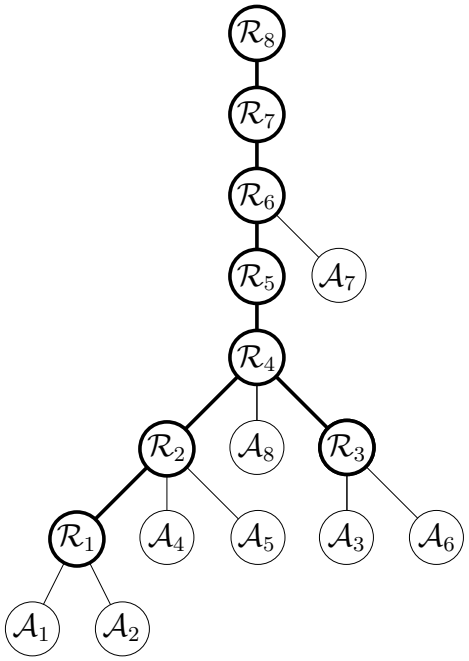
Note that in the second step, the nonzero pattern of the second pivot row ($\{4, 5, 6, 8\}$) is a loose upper bound. It cannot be attained by any one of the pivot row selections.

This process continues for subsequent steps. At the kth step of factorization, any row represented by a prior pivot row $\mathcal{R}_i$ (for $i < k$) will also be a candidate pivot row at step k if $k \in \mathcal{R}_i$. Any row i for which $k \in \mathcal{A}_i$ will also be a candidate pivot row at step k. Since fill-in causes all of these rows in $\mathbf{A}^{(k)}$ to take on the same upper bound nonzero pattern $\mathcal{R}_k$, all the sets $\{\mathcal{R}_i : k \in \mathcal{R}_i\}$ and $\{\mathcal{A}_i : k \in \mathcal{A}_i\}$ can be discarded. Thus, in the absence of column permutations,

$$\mathcal{R}_k = \left(\bigcup_{k=\min \mathcal{R}_i} \mathcal{R}_i \right) \cup \left(\bigcup_{k=\min \mathcal{A}_i} \mathcal{A}_i \right) \setminus \{k\}. \quad (1)$$

Fig. 1. Example matrix, factorization, and row-merge tree





We refer to the discarding of rows used to construct $\mathcal{R}_k$ using (1) as *regular row absorption*.

The set of remaining rows $\mathcal{R}_i$ and $\mathcal{A}_i$ that have not been absorbed define the upper bound nonzero pattern of the matrix $\mathbf{A}^{(k)}$. This bound is tighter than $\mathbf{L}_C$. Consider, for example, an dense upper Hessenberg matrix. It is strong Hall and $\mathbf{A}^\mathsf{T}\mathbf{A}$ is dense. The factor $\mathbf{L}_C$ is thus dense. However, our bound on each submatrix $\mathbf{A}^{(k)}$, as defined by the sets $\mathcal{R}_i$ and $\mathcal{A}_i$, is exact.

Equation 1 describes the row-merge tree, with $2n$ nodes. The tree has one node per original row $\mathcal{A}_i$ and one node per pivotal row $\mathcal{R}_i$. Pivotal nodes and edges between them are drawn with bold lines in Figure 1. Original node $\mathcal{A}_i$ represents the nonzero pattern of row i of $\mathbf{A}$. Its parent in the tree is the pivotal node $\min \mathcal{A}_i$. All original nodes are leaf nodes in the row-merge tree. Pivotal node $\mathcal{R}_k$ represents the nonzero pattern at step k of all candidate pivot rows at step k. These candidate pivot rows are all the rows i such that original node $\mathcal{A}_i$ is in the subtree rooted at pivotal node k, except for those rows from that set that have become pivotal by step k. The number of these pivotal rows selected from this set by step k is equal to the number of pivotal nodes in the subtree rooted at k. The parent of pivotal node k is $\min \mathcal{R}_k$.

The column elimination tree is a tree of n nodes. It is obtained by simply removing the n original nodes from the row-merge tree.

The row-merge tree exactly captures the size of the upper bound pattern of the LU factors, assuming arbitrary row permutations via partial pivoting, but with no column permutations. The upper bound of the off-diagonal nonzero pattern of row k of $\mathbf{U}$ is simply $\mathcal{R}_k$. The upper bound of the nonzero pattern of column k of $\mathbf{L}$ is represented implicitly. In terms of the unpermuted original row indices, it is the set of all original nodes in the subtree rooted at node k, and where one of these nodes is removed for each pivotal node in the subtree rooted at k. Which nodes are removed depends on the actual row permutation found during numerical factorization. Thus, the number of nonzeros in below the diagonal in column k of the upper bound of the pattern of $\mathbf{L}$ is equal to the number of original nodes in the subtree rooted at k minus the number of pivotal nodes in the subtree rooted at k. This is also the number of non-pivotal rows represented by $\mathcal{R}_k$. We denote it as l_k, and in the absence of column permutations it can be computed as

$$l_k = \left(\sum_{k=\min \mathcal{R}_i} l_i \right) + |\{i : k = \min \mathcal{A}_i\}| - 1. \tag{2}$$

For example, the second column of the upper bound of $\mathbf{L}$ consists of original rows 1, 2, 4, and 5, with one of these candidates removed at node $\mathcal{R}_1$ (either row 1 or 2) and another candidate row removed at node $\mathcal{R}_2$. There at most $l_2 = 2$ off-diagonal nonzeros in column 2 of the bound of $\mathbf{L}$, regardless of how the pivot rows are chosen. Similarly, there are at most $l_4 = 3$ off-diagonal nonzeros in column 4 of the bound of $\mathbf{L}$ (7 candidate pivot rows, minus 4 of them selected as pivot rows by step 4), and equivalently $\mathcal{R}_4$ represents the upper bound nonzero pattern of 3 non-pivotal rows.

A *super-row* is a row that represents a set of rows with the same nonzero pattern. Each pivotal row $\mathcal{R}_k$ is normally a super-row, since it represents all but one of

Algorithm 1: Simple symbolic LU factorization algorithm

Let $\mathcal{A}_i = \texttt{Struct}(\mathbf{A}_{i*})$ for all i
Let $\mathcal{R}_i = \emptyset$ for all i
elimination phase:
for $k = 1$ **to** n **do**
  find upper bound pattern of pivot row:
  $\mathcal{R}_k = \left(\bigcup_{k=\min \mathcal{R}_i} \mathcal{R}_i\right) \cup \left(\bigcup_{k=\min \mathcal{A}_i} \mathcal{A}_i\right) \setminus \{k\}$
  find upper bound number of nonzeros in pivot column:
  $l_k = \left(\sum_{k=\min \mathcal{R}_i} l_i\right) + |\{i : k = \min \mathcal{A}_i\}| - 1$
  regular row absorption:
  for each i such that $k = \min \mathcal{R}_i$ **do** $\mathcal{R}_i = \emptyset$ **end for**
  for each i such that $k = \min \mathcal{A}_i$ **do** $\mathcal{A}_i = \emptyset$ **end for**
  if $l_k = 0$ **then** $\mathcal{R}_k = \emptyset$ **end if**
end for

rows in the set of candidate pivot rows at step k, and this set is often of size larger than one. Original rows $\mathcal{A}_i$ are not super-rows. The number of rows represented by the super-row $\mathcal{R}_i$ is l_i. A *super-column* represents a set of columns with the same nonzero pattern. The idea of combining rows and columns with identical nonzero pattern was first introduced by Sherman [1975], who used super-columns in the context of sparse Cholesky factorization. Our algorithms use super-columns, but they do not appear in the algorithms or equations outlined here since they complicate the presentation. We use the notation $\|...\|$ to represent the sum of the sizes of the super-rows or super-columns in a set.

With just the sets $\mathcal{R}_i$ and $\mathcal{A}_i$ for all i, we have enough to efficiently compute a symbolic LU factorization in time proportional to the number of entries in the upper bound of $\mathbf{L}$ and $\mathbf{U}$, assuming no column permutations, as shown in Algorithm 1 [George and Ng 1987]. However, Algorithm 1 cannot be used as a basis for the COLAMD algorithm described in the next section because we need to permute the columns during symbolic factorization. With column permutations, computing (1) is more complicated, since we need to be able quickly find all those sets $\mathcal{R}_i$ and $\mathcal{A}_i$ that contain a given entry k. To keep track of this, we maintain a set $\mathcal{C}_j$ for each column j. The set $\mathcal{C}_j$ contains a reference to those sets $\mathcal{R}_i$ and $\mathcal{A}_i$ that contain entry j. Initially, $\mathcal{C}_j = \texttt{Struct}(\mathbf{A}_{*j})$, the nonzero pattern of column j. A bold entry $\mathbf{i}$ in $\mathcal{C}_j$ will refer to a pivotal set $\mathcal{R}_i$, and an entry i in $\mathcal{C}_j$ will refer to an original set $\mathcal{A}_i$. The terms $\mathbf{i}$ and i can both be contained in a set $\mathcal{C}_j$. In Figure 1, we have $\mathcal{C}_4 = \{\mathbf{2}, 6, 8\}$ after step 2. Rows 1, 4, and 5 do not appear in $\mathcal{C}_4$ since they have been used to construct prior pivot rows and have been discarded. If we define $\mathcal{C}_k = \{\mathbf{k}\}$ at the end of step k, then $\|\mathcal{C}_k\| = l_k$ is the number of nonzeros in column k of the bound of $\mathbf{L}$. If l_k is zero then $\mathcal{R}_k$ and $\mathcal{C}_k$ can be discarded. Pivot row $\mathcal{R}_k$ causes no fill-in, and does not represent any non-pivotal rows. This can occur for $k < n$ if the matrix is not strong Hall.

Algorithm 2 is a symbolic LU factorization algorithm that does not perform column permutations, but it has the required data structures required to do so (the sets $\mathcal{C}_j$ for all j). The initial storage required by Algorithm 2 is $O(|\mathbf{A}|)$. At step k,

Algorithm 2: Symbolic LU factorization algorithm (with $\mathcal{C}$'s)

Let $\mathcal{A}_i = \texttt{Struct}(\mathbf{A}_{i*})$ for all i
Let $\mathcal{C}_j = \texttt{Struct}(\mathbf{A}_{*j})$ for all j
Let $\mathcal{R}_i = \emptyset$ for all i
elimination phase:
for $k = 1$ **to** n **do**
  find upper bound pattern of pivot row:
  $\mathcal{R}_k = \left(\bigcup_{\mathbf{i}\in\mathcal{C}_k} \mathcal{R}_i\right) \cup \left(\bigcup_{i\in\mathcal{C}_k} \mathcal{A}_i\right) \setminus \{k\}$
  find upper bound number of nonzeros in pivot column:
  $l_k = \left(\sum_{\mathbf{i}\in\mathcal{C}_k} l_i\right) + |\{i : i \in \mathcal{C}_k\}| - 1$
  regular row absorption:
  for each i $\in \mathcal{C}_k$ **do** $\mathcal{R}_i = \emptyset$ **end for**
  for each $i \in \mathcal{C}_k$ **do** $\mathcal{A}_i = \emptyset$ **end for**
  symbolic update:
  for each column $j \in \mathcal{R}_k$ **do**
    $\mathcal{C}_j = \mathcal{C}_j \setminus \mathcal{C}_k$
  end for
  $K = \{\mathbf{k}\}$
  if $l_k = 0$ **then** $\mathcal{R}_k = \emptyset$; $K = \emptyset$ **end if**
  for each column $j \in \mathcal{R}_k$ **do**
    $\mathcal{C}_j = \mathcal{C}_j \cup K$
  end for
  $\mathcal{C}_k = K$
end for

$$|\mathcal{R}_k| < \sum_{\mathbf{i}\in\mathcal{C}_k} |\mathcal{R}_i| + \sum_{i\in\mathcal{C}_k} |\mathcal{A}_i|$$

because $\mathcal{R}_k$ is computed using (1). The sets used to construct $\mathcal{R}_k$ are then discarded. Thus, at each step, the total storage required for the sets $\mathcal{R}_i$ for all i does not increase. When the outer product is formed, each column $\mathcal{C}_j$ reduces in size or stays the same size. If $j \in \mathcal{R}_k$, then from (1) this implies $\exists i$ such that $j \in \mathcal{R}_i$ or $j \in \mathcal{A}_i$. By definition, $j \in \mathcal{R}_i$ or $j \in \mathcal{A}_i$ implies that $i \in \mathcal{C}_j$. This means the sets $\mathcal{C}_k$ and $\mathcal{C}_j$ have a non-empty intersection. Since $\mathcal{C}_j$ is replaced with $(\mathcal{C}_j \setminus \mathcal{C}_k) \cup \{\mathbf{k}\}$, it cannot increase in size. Thus, each step reduces or maintains the total storage required for the sets $\mathcal{A}_i$ and $\mathcal{R}_i$ for all i and the sets $\mathcal{C}_j$ for all j.

The total asymptotic time taken is dominated by the symbolic update. Computing the pivot rows $\mathcal{R}_k$ for the entire algorithm takes time proportional to the size of the upper bound pattern of $\mathbf{U}$, since $\mathcal{R}_k$ is the upper bound pattern of the off-diagonal part of row k of $\mathbf{U}$, and since $\mathcal{R}_k$ is discarded when it is included in a set union for a subsequent pivot row. Modifying a single column $\mathcal{C}_j$ in the symbolic update takes no more than $O(|\mathbf{A}_{*j}|)$ time, since $|\mathcal{C}_j| \leq |\mathbf{A}_{*j}|$. Column j is modified at most once for each row $\mathcal{R}_k$ that contains column index j. Thus, column j is modified at most υ_j times during the entire algorithm, where we define υ_j as the number of entries in the upper bound pattern on column j of the matrix $\mathbf{U}$ for any

row permutation $\mathbf{P}$ due to partial pivoting. Thus, the total time taken is

$$O\left(\sum_{j=1}^{n} |\mathbf{A}_{*j}| v_j\right).$$

This is the same time required to compute the sparse matrix product $\mathbf{A}$ times the upper bound of the pattern of $\mathbf{U}^{\mathsf{T}}$, and is typically much less than the time required for numerical factorization, which is equal to the time required to compute the sparse matrix product $\mathbf{L}$ times $\mathbf{U}$ [Gustavson 1978],

$$O\left(\sum_{k=1}^{n} |\mathbf{L}_{*k}||\mathbf{U}_{k*}|\right).$$

4. THE COLAMD ORDERING ALGORITHM

We now present a column ordering method based on the symbolic LU factorization algorithm presented in the last section and with the same time complexity and storage requirements. At step k, we select a pivot column c to minimize some metric on all of the candidate pivot columns as an attempt to reduce fill-in. Columns c and k are then exchanged.

The presence of dense (or nearly dense) rows or columns in $\mathbf{A}$ can greatly affect both the ordering quality and run time. A single dense row in $\mathbf{A}$ renders all our bounds useless, unless dense rows are treated differently. If there is one completely dense row, and that row is not treated differently, the upper bound on the nonzero pattern of $\mathbf{A}^{(1)}$ is a completely dense matrix. We can hope that this dense row will not be selected as the first pivot row during numerical factorization, and withhold the row from the ordering algorithm. Dense columns do not affect the ordering quality, but they do increase the ordering time. A single dense column increases the ordering time by $O(n^2)$. Dense columns are withheld from the symbolic factorization and ordering algorithm, and placed last in the column ordering $\mathbf{Q}$. Determining how dense a row or column should be for it to be withheld is problem dependent. We used the same default threshold used by MATLAB's COLMMD, 50%, which is probably too high for most matrices.

Taking advantage of super-columns can greatly reduce the ordering time. In the symbolic update step, we look at all columns j in the set $\mathcal{R}_k$. If any two or more columns have the same pattern (tested via a hash function similar to the one used in [Ashcraft 1995]), they are merged into a single super-column. This saves both time and storage in subsequent steps. Selecting a super-column allows us to *mass-eliminate* [George and McIntyre 1978] all columns represented by the super-column, and we skip ahead the same number of steps in the symbolic factorization. If the pattern of column or super-column $j \in \mathcal{R}_k$ becomes the same as the pivot column pattern ($\mathcal{C}_j = \{\mathbf{k}\}$) after the symbolic update, it can be eliminated immediately. As elimination progresses, the nonzero pattern of all the columns represented by a super-column change in an identical manner. Thus, we only need to maintain the degree, or other column selection metric, for the representative column.

By necessity, the choice of the pivot column is a heuristic, since obtaining an ordering with minimum fill-in is an NP-complete problem [Yannakakis 1981]. Several strategies are possible. Each swaps a column c with column k that minimizes some

metric on the new kth column, described below. For most of these methods, we need to compute the initial metric for each column prior to symbolic factorization and ordering, and then recompute the metric for each column j in the pivot row pattern $\mathcal{R}_k$ at step k. A generic ordering algorithm is based on algorithm 1, the symbolic LU factorization. At the beginning of the **for** loop, it uses one of the following metrics (described in Sections 4.1 through 4.6) to select a column c, and then interchanges columns k and c. Each iteration also updates the metric, which is often the most costly part of the computation.

4.1 COLMMD approximate external row degree

The COLMMD column ordering algorithm in MATLAB [Gilbert et al. 1992] selects as pivot the column that minimizes a loose upper bound on the external row degree,

$$\|\mathcal{R}_k\| \leq \sum_{\mathbf{i}\in\mathcal{C}_k} (\|\mathcal{R}_i \setminus \{k\}\|) + \sum_{i\in\mathcal{C}_k} (\|\mathcal{A}_i \setminus \{k\}\|). \tag{3}$$

Here, and in the following sections, $\|\mathcal{R}_k\|$ refers to the metric for pivot column k once it is exchanged with the candidate column c. We thus need to know $\|\mathcal{R}_c\|$ for all candidate columns c. Note that in (3), $k \in \mathcal{R}_i$, so $\|\mathcal{R}_i \setminus \{k\}\| = \|\mathcal{R}_i\| - 1$. If super-columns are present, this expression becomes $\|\mathcal{R}_i\|$ minus the number of columns represented by super-column k. Using this bound for the symbolic update does not increase the asymptotic time complexity of the ordering algorithm above that of the symbolic LU factorization algorithm. Computing the COLMMD metric for the initial columns of $\mathbf{A}$ is very fast, taking only $O(|\mathbf{A}|)$ time.

4.2 Exact external row degree

This method selects the pivot column to minimize the size of the resulting pivot row,

$$\|\mathcal{R}_k\| = \left\| \left(\bigcup_{\mathbf{i}\in\mathcal{C}_k} \mathcal{R}_i \right) \cup \left(\bigcup_{i\in\mathcal{C}_k} \mathcal{A}_i \right) \setminus \{k\} \right\|.$$

(We exclude the pivot column itself, thus computing an *external* row degree [Liu 1985].) The exact degree is expensive to compute, and our experience with symmetric orderings is that exact degrees do not provide better orderings than good approximate degrees (such as AMD-based approximations) [Amestoy et al. 1996; 20xx]. Selecting `spparms ('tight')` in MATLAB improves the ordering computed by COLMMD by forcing it to use an exact row degree, but at high cost in ordering time (it can take more time than the numerical factorization). It is not a practical method for most purposes. We thus did not test this method.

4.3 AMD approximate external row degree

The AMD algorithm for ordering symmetric matrices prior to a Cholesky factorization is based on a bound on the external row degree that is tighter than the COLMMD bound [Amestoy et al. 1996; 20xx]. It was first used in the unsymmetric-pattern multifrontal method (UMFPACK), to compute the ordering during numerical factorization [Davis and Duff 1997; 1999]. In the context of a column ordering

algorithm for the sparse partial pivoting method, the analogous bound on $\|\mathcal{R}_k\|$ is

$$\|\mathcal{R}_k\| \leq \|\mathcal{R}_s \setminus \{k\}\| + \sum_{\mathbf{i} \in \mathcal{C}_k \setminus \{s\}} (\|\mathcal{R}_i \setminus \mathcal{R}_s\|) + \sum_{i \in \mathcal{C}_k} (\|\mathcal{A}_i \setminus \mathcal{R}_s\|), \tag{4}$$

where $\mathcal{R}_s$ is the most recent pivot row that modified $\mathcal{C}_k$ in a prior symbolic update (thus $\mathbf{s} \in \mathcal{C}_k$).

To compute the initial metric, we first sort the rows of $\mathbf{A}$ according to the number of nonzeros in each row, with denser rows first. We then form the matrix

$$\begin{bmatrix} \mathbf{I} & \mathbf{SA} \\ \mathbf{I} & 0 \end{bmatrix}$$

where $\mathbf{S}$ is the row sort permutation. We then perform n steps of elimination with no column permutations to form the representation of the Schur complement. The approximate external row degree (discussed below) is computed during this initialization phase. The result is an approximation of the external row degrees of each row of $\mathbf{A}^\mathsf{T}\mathbf{A}$.

The time to compute the initial AMD metric is no more than the time it takes to compute the sparse matrix product $\mathbf{A}\mathbf{A}^\mathsf{T}$, but this bound on the time is loose. The time can be much less because sparser rows may be absorbed by denser rows, and the initial elimination for the sparser rows can be skipped. Computing exact initial degrees would take time proportional to computing $\mathbf{A}^\mathsf{T}\mathbf{A}$, and we would not be able to take advantage of row absorption. Note that our final COLAMD algorithm does not use the initial AMD metric, however. It uses a faster method that generates better orderings instead.

After the initialization phase is done, the elimination phase performs the symbolic LU factorization and finds the column ordering. After the kth pivot column is selected, the AMD approximation must be recomputed for each column j in the pivot row pattern. The time to compute the AMD approximation during the elimination phase is asymptotically the same as computing the COLMMD bound, but the constant is higher by a factor of 2, since it is computed in two phases. The first phase computes all the required set differences, and the second phase adds up the set differences for each column in the pivot row pattern. Algorithm 3 is a generic COLAMD algorithm, with the initial metric unspecified but with the AMD metric used during the elimination phase. To simplify the presentation, we assume there are no super-columns. It is identical to Algorithm 2, except that column permutations are performed and a column metric d_j for column j is maintained. All entries in the work arrays v and w of size n are initially less than a tag value, t. This is restored at the end of the kth iteration by increasing the value of t. At the end of the first phase, $w_i - t$ is the size of the set difference, $\|\mathcal{R}_i \setminus \mathcal{R}_k\|$, and $v_i - t$ is equal to $\|\mathcal{A}_i \setminus \mathcal{R}_k\|$.

4.4 Size of the Householder update

The symbolic LU factorization also computes the pattern of $\mathbf{R}$ for a QR factorization of $\mathbf{A}$ [Coleman et al. 1986; Hare et al. 1993]. At step k, the size of the Householder rectangular update is $\|\mathcal{C}_k\|$-by-$\|\mathcal{R}_k\|$. The term $\|\mathcal{C}_k\| = l_k$ is known exactly; the $\|\mathcal{R}_k\|$ term can be computed or approximated using any of the above methods. The method we tested selected the pivot column to minimize the product

Algorithm 3: Generic COLAMD algorithm

Let $\mathcal{A}_i = \texttt{Struct}(\mathbf{A}_{i*})$ for all i
Let $\mathcal{C}_j = \texttt{Struct}(\mathbf{A}_{*j})$ for all j
Let $\mathcal{R}_i = \emptyset$ for all i
compute initial metric d_j for each column j
$w = 0$; $v = 0$; $t = 1$
elimination phase:
for $k = 1$ **to** n **do**
  select pivot column c that minimizes d_c and swap columns c and k
  find upper bound pattern of pivot row:
  $\mathcal{R}_k = \left(\bigcup_{\mathbf{i}\in\mathcal{C}_k} \mathcal{R}_i\right) \cup \left(\bigcup_{i\in\mathcal{C}_k} \mathcal{A}_i\right) \setminus \{k\}$
  find upper bound number of nonzeros in pivot column:
  $l_k = \left(\sum_{\mathbf{i}\in\mathcal{C}_k} l_i\right) + |\{i : i \in \mathcal{C}_k\}| - 1$
  regular row absorption:
  for each i $\in \mathcal{C}_k$ **do** $\mathcal{R}_i = \emptyset$ **end for**
  for each $i \in \mathcal{C}_k$ **do** $\mathcal{A}_i = \emptyset$ **end for**
  symbolic update and approximate degree computation:
  for each column $j \in \mathcal{R}_k$ **do**
    $\mathcal{C}_j = \mathcal{C}_j \setminus \mathcal{C}_k$
    for each pivotal row **i** $\in \mathcal{C}_j$ **do**
      if $w_i < t$ **then** $w_i = \|\mathcal{R}_i\| + t$
      $w_i = w_i - 1$
    end for
    for each original row $i \in \mathcal{C}_j$ **do**
      if $v_i < t$ **then** $v_i = \|\mathcal{A}_i\| + t$
      $v_i = v_i - 1$
    end for
  end for
  $K = \{\mathbf{k}\}$
  if $l_k = 0$ **then** $\mathcal{R}_k = \emptyset$; $K = \emptyset$ **end if**
  for each column $j \in \mathcal{R}_k$ **do**
    $d_j = \|\mathcal{R}_k \setminus \{k\}\|$
    for each pivotal row **i** $\in \mathcal{C}_j$ **do** $d_j = d_j + (w_i - t)$ **end for**
    for each original row $i \in \mathcal{C}_j$ **do** $d_j = d_j + (v_i - t)$ **end for**
    $\mathcal{C}_j = \mathcal{C}_j \cup K$
  end for
  $\mathcal{C}_k = K$
  $t = t + \max(\max_i \|\mathcal{R}_i\|, \max_i \|\mathcal{A}_i\|)$
end for

of $\|\mathcal{C}_k\|$ and the AMD approximation of $\|\mathcal{R}_k\|$ in (4). We tested and then discarded this method, since it gave much worse orderings than the other methods.

4.5 Approximate Markowitz criterion

The Markowitz heuristic selects as pivot the entry a_{ij} that minimizes the product of the degrees of row i and column j, which is the amount of work required in the subsequent outer product step [Markowitz 1957]. The criterion assumes that the

first $k-1$ pivot rows and columns have been selected. In our case, we do not know the exact pivot row ordering yet, so we do not know the true row or column degrees. In our tests, we selected the pivot column to minimize

$$\|\mathcal{C}_k\| \cdot \max \left(\max_{\mathbf{i} \in \mathcal{C}_k} \|\mathcal{R}_i\|, \max_{i \in \mathcal{C}_k} \|\mathcal{A}_i\| \right) \tag{5}$$

Compared with the AMD approximation described in Section 4.3, this is a tighter upper bound on the actual pivot row degree if row i is selected as the k-th pivot row. Although the largest set $\mathcal{R}_i$ or $\mathcal{A}_i$ for all i in the set $\mathcal{C}_k$ does not bound the pattern $\mathcal{R}_k$ of the k-th pivot row for arbitrary row partial pivoting, it does bound the size of the actual pivot row once the row ordering is selected. Since all of the terms used in (5) are known exactly, no approximations need to be used. We tested and then discarded this method, since it gave much worse orderings than the other methods.

4.6 Approximate deficiency

The *deficiency* of a pivot is the number of new nonzero entries that would be created if that pivot is selected; selecting the pivot with least deficiency leads to a minimum deficiency ordering algorithm. Exact deficiency is very costly to compute. Approximate minimum deficiency ordering algorithms have been successfully used for symmetric matrices, in the context of Cholesky factorization [Ng and Raghavan 1999; Rothberg and Eisenstat 1998]. In a nonsymmetric context, the deficiency of column k can be bounded by

$$\|\mathcal{C}_k\|\|\mathcal{R}_k\| - \sum_{\mathbf{i} \in \mathcal{C}_k} l_i(\|\mathcal{R}_i\| - 1) - \sum_{i \in \mathcal{C}_k} (\|\mathcal{A}_i\| - 1)$$

Any new nonzeros are limited to the $\|\mathcal{C}_k\|$-by-$\|\mathcal{R}_k\|$ Householder update. Each super-row $i \in \mathcal{C}_k$, however, is contained in this submatrix and thus reduces the possible fill-in by the number of nonzeros it represents. The $\|\mathcal{C}_k\|$ and $\|\mathcal{R}_i\|$ terms are known exactly; we used the AMD approximation for $\|\mathcal{R}_k\|$, from (4). Even if $|\mathcal{R}_k|$ were known exactly for each candidate column, we would still have an approximation of the exact deficiency.

We tested one variant of approximate deficiency [Kern 1999; Larimore 1998]. The initialization phase computes the approximate deficiency of all columns, and the approximate deficiency is recomputed any time a column is modified. The experimental results were mixed. It took more time to compute the ordering, and the overall ordering quality for our entire test suite was about the same as our final COLAMD algorithm. Finding a good approximate deficiency metric for unsymmetric matrices is an open question.

4.7 Aggressive row absorption

As a by-product of the AMD row degree computation, we compute the sizes of the set differences $\|\mathcal{R}_i \setminus \mathcal{R}_k\|$ when super-row $\mathcal{R}_k$ is the bound on the pivot row at step k, for all rows $i \in \mathcal{C}_k$. If we find that this set difference is zero, $\mathcal{R}_i$ is a subset of row $\mathcal{R}_k$. The presence of row i does not affect the exact row degree of any column, although it does affect the COLMMD and AMD approximations. Row i can be absorbed into $\mathcal{R}_k$. We refer to the deletion of $\mathcal{R}_i$ when $\mathbf{i} \notin \mathcal{C}_k$, or the deletion of

$\mathcal{A}_i$ when $i \notin \mathcal{C}_k$ as *aggressive row absorption*. Regular row absorption occurs when $\mathcal{R}_i$ is absorbed into row $\mathcal{R}_k$ when $\mathbf{i} \in \mathcal{C}_k$, or when $\mathcal{A}_i$ is absorbed when $i \in \mathcal{C}_k$. Regular row absorption is similar to *element absorption*, introduced by Duff and Reid [Duff and Reid 1983]. If AMD row degrees are computed as the initial metric, then *initial* aggressive row absorption can occur in that phase as well. Aggressive row absorption reduces the run time, since it costs almost nothing to detect this condition when the AMD metric is used, and results in fewer rows to consider in subsequent steps. It results in a tighter COLMMD or AMD approximation to the row degree.

4.8 COLAMD variants

We tested sixteen variants of COLAMD, based on all possible combinations of the following design decisions:

(1) Initial metric: COLMMD or AMD approximation.
(2) Metric computed during the symbolic update: COLMMD or AMD approximation.
(3) With or without initial aggressive row absorption.
(4) With or without aggressive row absorption during the elimination phase.

Note that some of the combinations are costly. Aggressive row absorption is easy when using the AMD row degrees, but difficult when using the COLMMD approximation. Details of our experimental results on these variants are given in Larimore's thesis [1998].

Since the AMD metric was shown to be superior to the COLMMD approximation in the context of minimum degree orderings for sparse Cholesky factorization [Amestoy et al. 1996; 20xx], we expected the four variants based solely on the AMD metric to be superior, so much so that we did not intend to test all sixteen methods. To our surprise, we found that better orderings were obtained with an initial COLMMD metric and an AMD metric during the elimination phase. We discovered this by accident. A bug in our initial computation of the AMD metric resulted in the COLMMD approximation being computed instead. This "bug" gave us better orderings, so we kept it. Using an initial COLMMD metric gave, on average, orderings with 8% fewer floating-point operations in the subsequent numerical factorization than using an initial AMD metric. The initial COLMMD metric is also faster to compute.

The version of our ordering algorithm that we recommend, which we now simply call COLAMD, uses the initial COLMMD metric, the AMD metric for the symbolic update (described in Section 4.3), no initial aggressive row absorption, super-rows, super-columns, and aggressive row absorption during the elimination phase.

MATLAB's COLMMD ordering algorithm uses the COLMMD metric throughout, multiple elimination with a relaxed threshold [Liu 1985], super-rows, super-columns, mass elimination, and aggressive row absorption. Aggressive row absorption is not free, as it is in COLAMD. Instead, an explicit test is made every three stages of multiple elimination. Super-columns are searched for every three stages, by default. Rows more than 50% dense are withheld. Since the COLMMD metric can lead to lower quality orderings than an exact metric, options are provided for

selecting an exact external row degree metric, modifying the multiple elimination threshold, changing the frequency of super-column detection and aggressive row absorption, and changing the dense row threshold. Selecting `spparms ('tight')` in MATLAB improves the ordering computed by COLMMD, but at high cost in ordering time. The default options use the COLMMD approximation described in Section 4.1. We do not use the `'tight'` option in COLMMD.

5. EXPERIMENTAL RESULTS

We tested our COLAMD ordering algorithm with three sets of matrices: square nonsymmetric matrices, rectangular matrices, and symmetric positive definite matrices. Our test set is taken from the University of Florida sparse matrix collection [Davis 2000], which includes the Harwell/Boeing test set [Duff et al. 1989; 1992], the linear programming problems in Netlib at `http://www.netlib.org` [Dongarra and Grosse 1987], as well as many other matrices. We include all matrices in our tests except for complex matrices, nonsymmetric matrices for which only the pattern was provided, and unassembled finite element matrices. Some matrices include explicit zero entries in the description of their pattern. Since we ignore numerical cancellation, we included these entries when finding the ordering and determining the resulting factorization. Each method is compared by ordering time and quality (number of nonzeros in the factors, and number of floating-point operations to compute the factorization). Although we tested nearly all matrices in the collection, we present here a summary of only some of the larger problems (those for which the best ordering resulted in a factorization requiring 10^7 operations or more to compute). Complete results are presented in Larimore's thesis [1998], available as a technical report at `http://www.cise.ufl.edu/`. We performed these experiments on a Sun Ultra Enterprise 4000/5000 with 2 GB of main memory and eight 248 Mhz UltraSparc-II processors (only one processor was used). The code was written in ANSI/ISO C and compiled using the Solaris C compiler (via Mathwork's `mex` shell script for interfacing MATLAB to software written in C), with strict ANSI/ISO compliance.

5.1 Square nonsymmetric matrices

For square nonsymmetric matrices, we used COLAMD to find a column preordering $\mathbf{Q}$ for sparse partial pivoting. These results are compared with COLMMD with default option settings, and with AMDBAR [Amestoy et al. 1996; 20xx] applied to the pattern of $\mathbf{A}^{\mathsf{T}}\mathbf{A}$ (ignoring numerical cancellation, and withholding the same dense rows and columns from $\mathbf{A}$ that were withheld by COLAMD and COLMMD). The AMDBAR routine is the same as MC47BD in the Harwell Subroutine Library, except that it does not perform aggressive row absorption. Both AMDBAR and MC47BD use the AMD-style approximate degree. For sparse partial pivoting with the matrices in our test set, AMDBAR provides slightly better orderings than MC47BD. After finding a column ordering, we factorized the matrix $\mathbf{AQ}$ with the SuperLU package [Demmel et al. 1999]. This package was chosen since COLAMD was written to replace the column ordering in the current version of SuperLU.

SuperLU is based on the BLAS [Dongarra et al. 1990].[2]

We excluded matrices that can be permuted to upper block triangular form [Duff 1981] with a substantial improvement in factorization time, for two reasons: (1) our bounds are not tight if the matrix $\mathbf{A}$ is not strong Hall, and (2) SuperLU does not take advantage of reducibility to block upper triangular form. Such matrices should be factorized by ordering and factorizing each irreducible diagonal submatrix, after permuting the matrix to block triangular form. Our resulting test set had 106 matrices, all requiring more than 10^7 or more operations to factorize.

A representative sample is shown in Table I, sorted according to COLAMD versus COLMMD ordering quality.[3] Table II reports the ordering time of COLAMD, COLMMD, and AMDBAR (the time to compute the pattern of $\mathbf{A}^\mathsf{T}\mathbf{A}$ is included in the AMDBAR time). For comparison, the last column reports the SuperLU factorization time, using the COLAMD ordering. Table III gives the resulting number of nonzeros in $\mathbf{L}+\mathbf{U}$, and the floating-point operations required to factorize the permuted matrix, for each of the ordering methods. The median results reported in this paper are for just the representative samples, not the full test sets. The results for the samples and the full test sets are similar.

COLAMD is superior to the other methods for these matrices. COLAMD was typically 3.9 times faster than COLMMD and 2.1 times faster than AMDBAR. For two matrices (LHR17C and AF23560), COLMMD took more time to order the matrix than SuperLU took to factorize it. The orderings found by COLMMD result in a median increase of 10% in nonzeros in the LU factors and 36% in floating-point operations, as compared to COLAMD. The ordering quality of COLAMD and AMDBAR are similar, although there are large variations in both directions in a small number of matrices. For a few matrices (in our larger test set, not in Table I) AMDBAR requires more space to store $\mathbf{A}^\mathsf{T}\mathbf{A}$ than SuperLU requires to factorize the permuted matrix.

5.2 Rectangular matrices

For m-by-n rectangular matrices with $m > n$, we found a column ordering $\mathbf{Q}$ for the Cholesky factorization of $(\mathbf{AQ})^\mathsf{T}(\mathbf{AQ})$, which is one method for solving a least squares problem. If $m < n$, we found a row ordering $\mathbf{P}$ for the Cholesky factorization of $(\mathbf{PA})\mathbf{D}^2(\mathbf{PA})^\mathsf{T}$, which arises in interior point methods for solving linear programming problems [Karmarkar 1978; Wright 1996]. Here, $\mathbf{D}$ is a diagonal matrix. In the latter case, COLAMD and COLMMD found a column ordering of $\mathbf{A}^\mathsf{T}$ and we used that as the row ordering $\mathbf{P}$. We compared these two methods with AMDBAR on the corresponding matrix, $\mathbf{A}^\mathsf{T}\mathbf{A}$ (if $m > n$) or $\mathbf{A}\mathbf{A}^\mathsf{T}$ (if $m < n$).

Our test set was limited. It included only 37 matrices requiring more than 10^7 operations; all but three of these were Netlib linear programming problems (with $m < n$). A representative selection[4] is shown in Table IV.

[2]We used the BLAS routines provided in SuperLU because factorization time is a secondary measure, and because we had difficulty using the Sun-optimized BLAS from a MATLAB-callable driver.

[3]To obtain the sample, we sorted the 106 matrices according to the relative floating-point operation count for the COLAMD ordering and the COLMMD ordering, and then selected every 8th matrix. We also included TWOTONE, the matrix with the largest dimension in the test set.

[4]Every 5th matrix from the 37 large matrices, in increasing order of COLAMD versus COLMMD

Table I. Unsymmetric matrices

matrix	n	$\|\mathbf{A}\|$	description
GOODWIN	7320	324784	finite-element, fluid dynamics, Navier-Stokes and elliptic mesh (R. Goodwin)
RAEFSKY2	3242	294276	incompressible flow in pressure-driven pipe (A. Raefsky)
TWOTONE	120750	1224224	frequency-domain analysis of nonlinear analog circuit [Feldmann et al. 1996]
RMA10	46835	2374001	3D computational fluid dynamics (CFD), Charleston Harbor (Steve Bova)
LHR17C	17576	381975	light hydrocarbon recovery problem [Zitney et al. 1996]
AF23560	23560	484256	airfoil [Bai et al. 1996]
EPB2	25228	175027	plate-fin heat exchanger with flow redistribution (D. Averous)
RDIST3A	2398	61896	chemical process separation [Zitney 1992]
GARON2	13535	390607	2D finite-element, Navier-Stokes (A. Garon)
GRAHAM1	9035	335504	Galerkin finite-element, Navier-Stokes, two-phase fluid flow (D. Graham)
CAVITY25	4562	138187	driven cavity, finite-element (A. Chapman)
EX20	2203	69981	2D attenuation of a surface disturbance, nonlinear CFD (Y. Saad)
WANG1	2903	19093	electron continuity, 3D diode [Miller and S. 1991]
EX14	3251	66775	2D isothermal seepage flow (Y. Saad)
EX40	7740	458012	3D die swell problem on a square die, computational fluid dynamics (Y. Saad)

We compared the three methods based on their ordering time, number of nonzeros in the factor $\mathbf{L}$, and floating-point operation count required for the Cholesky factorization. These metrics were obtained from SYMBFACT in MATLAB, a fast symbolic factorization [George and Liu 1980; 1981; Gilbert et al. 1994]. We did not perform the numerical Cholesky factorization. The time to construct $\mathbf{A}^\mathsf{T}\mathbf{A}$ or $\mathbf{A}\mathbf{A}^\mathsf{T}$ is included in the ordering time for AMDBAR. The results are shown in Tables V and VI.

For these matrices, COLAMD was twice as fast as COLMMD and slightly faster than AMDBAR. All three produced comparable orderings. Each method has a few matrices it does well on, and a few that it does poorly on. COLAMD and COLMMD both typically require less storage than AMDBAR, sometimes by several orders of magnitude, because of the need to compute the pattern of $\mathbf{A}^\mathsf{T}\mathbf{A}$ or $\mathbf{A}\mathbf{A}^\mathsf{T}$ prior to ordering it with AMDBAR. The size of the Cholesky factors always dominates the size of $\mathbf{A}^\mathsf{T}\mathbf{A}$ or $\mathbf{A}\mathbf{A}^\mathsf{T}$, however. The primary advantage of COLAMD and COLMMD over AMDBAR in this context is the ability to analyze the Cholesky factorization by finding the ordering and size of the resulting factor $\mathbf{L}$ in space proportional to the number of nonzeros in the matrix $\mathbf{A}$, rather than proportional to the size of the matrix to be factorized ($\mathbf{A}^\mathsf{T}\mathbf{A}$ or $\mathbf{A}\mathbf{A}^\mathsf{T}$).

ordering quality, was chosen.

Table II. Ordering and factorization time, in seconds

matrix	COLAMD	COLMMD	AMDBAR	SuperLU
GOODWIN	0.31	0.42	1.51	44.65
RAEFSKY2	1.11	1.58	2.11	62.36
TWOTONE	5.88	70.82	14.15	306.85
RMA10	2.37	25.71	12.07	120.67
LHR17C	1.62	15.22	3.08	5.80
AF23560	6.64	115.99	1.72	108.35
EPB2	0.44	2.62	0.61	16.32
RDIST3A	0.03	0.07	0.21	0.44
GARON2	0.61	1.19	1.29	25.12
GRAHAM1	0.43	0.58	1.50	31.63
CAVITY25	0.12	0.21	0.46	2.95
EX20	0.08	0.17	0.24	1.13
WANG1	0.07	0.27	0.08	2.65
EX14	0.10	0.53	0.20	2.47
EX40	2.34	13.58	2.61	27.21
Median time relative to COLAMD	-	3.9	2.1	37.1

Table III. Ordering quality, as factorized by SuperLU

matrix	Nonzeros in $\mathbf{L}+\mathbf{U}$ (10^3)			Flop. count (10^6)		
	COL-AMD	COL-MMD	AMD-BAR	COL-AMD	COL-MMD	AMD-BAR
GOODWIN	5634	3103	2734	1909	665	498
RAEFSKY2	3684	3236	2877	2374	2012	1682
TWOTONE	21030	19624	21280	8848	8923	8578
RMA10	18540	17544	15624	5064	5303	4041
LHR17C	1715	1767	1717	111	120	112
AF23560	12110	13333	13131	4515	5350	5052
EPB2	3186	3547	3135	646	839	620
RDIST3A	233	264	231	11	15	11
GARON2	5179	5120	5128	1061	1563	1505
GRAHAM1	4920	5344	4518	1333	2115	1414
CAVITY25	1146	1347	1210	130	219	178
EX20	483	589	464	48	86	50
WANG1	632	868	752	120	242	174
EX14	711	1012	885	91	214	157
EX40	4315	8537	6072	1075	5369	2606
Median result relative to COLAMD	-	1.10	0.99	-	1.36	1.05

Table IV. Rectangular matrices (all linear programming problems)

matrix	m	n	$\|\mathbf{A}\|$	$\|\mathbf{AA}^\mathsf{T}\|$ or $\|\mathbf{A}^\mathsf{T}\mathbf{A}\|$	description
GRAN	2629	2525	20111	60511	British Petroleum operations
NUG08	912	1632	7296	28816	LP lower bound for a quadratic assignment problem [Resende et al. 1995]
FIT1P	627	1677	9868	274963	fitting linear inequalities to data, min. sum of piecewise-linear penalties (R. Fourer)
KLEIN2	477	531	5062	139985	(E. Klotz)
PILOT87	2030	6680	74949	238624	(J. Stone)
PDS_20	33874	108175	232647	320196	military airlift operations [Carolan et al. 1990]
D2Q06C	2171	5831	33081	56153	(J. Tomlin)

Table V. Ordering time, in seconds

matrix	COLAMD	COLMMD	AMDBAR
GRAN	0.04	0.07	0.06
NUG08	0.05	0.06	0.04
FIT1P	0.01	0.01	0.08
KLEIN2	0.01	0.02	0.07
PILOT87	0.56	2.60	0.52
PDS_20	3.86	10.53	3.92
D2Q06C	0.07	0.44	0.08
Median time relative to COLAMD	-	2.00	1.14

Table VI. Ordering quality

matrix	Nonzeros in $\mathbf{L}$ (10^3)			Flop. count (10^6)		
	COL-AMD	COL-MMD	AMD-BAR	COL-AMD	COL-MMD	AMD-BAR
GRAN	191	158	143	48	33	27
NUG08	210	202	230	78	70	92
FIT1P	197	197	197	82	82	82
KLEIN2	81	82	80	17	18	17
PILOT87	423	475	406	169	195	164
PDS_20	6827	8159	6929	8598	12640	8812
D2Q06C	175	271	147	35	97	27
Median result relative to COLAMD	-	1.00	0.99	-	1.02	0.98

Table VII. Symmetric matrices

matrix	n	$\|\mathbf{A}\|$	description
PWT	36519	326107	pressurized wind tunnel (R. Grimes)
MSC00726	726	34518	symmetric test matrix from MSC/NASTRAN
BCSSTK38	8032	355460	stiffness matrix, airplane engine component (R. Grimes)
NASA2146	2146	72250	structural engineering problem (NASA Langley)
CFD2	123440	3087898	computational fluid dynamics (E. Rothberg)
3DTUBE	45330	3213618	3D pressure tube (E. Rothberg)
GEARBOX	153746	9080404	ZF aircraft flap actuator (E. Rothberg)
CRYSTM02	13965	322905	crystal, free vibration mass matrix (R. Grimes)
FINAN512	74752	596992	portfolio optimization [Berger et al. 1995]

Table VIII. Ordering time, in seconds

matrix	SYMAMD	SYMMMD	AMDBAR	MC47BD
PWT	0.78	3.53	0.45	0.44
MSC00726	0.05	0.24	0.02	0.02
BCSSTK38	0.53	22.45	0.12	0.11
NASA2146	0.08	0.49	0.02	0.02
CFD2	7.36	48.68	3.90	3.71
3DTUBE	6.09	77.75	0.95	0.92
GEARBOX	16.58	454.49	2.94	2.89
CRYSTM02	0.71	1.36	0.28	0.28
FINAN512	1.60	2.61	0.92	0.95
Median time relative to SYMAMD	-	6.13	0.39	0.39

5.3 Symmetric matrices

For symmetric matrices, MATLAB's SYMMMD routine constructs a matrix $\mathbf{M}$ such that the pattern of $\mathbf{M}^{\mathsf{T}}\mathbf{M}$ is the same as $\mathbf{A}$, and then finds a column ordering of $\mathbf{M}$ using COLMMD. There is one row in $\mathbf{M}$ for each entry a_{ij} below the diagonal of $\mathbf{A}$, with nonzero entries in column i and j. This method gives a reasonable ordering for the Cholesky factorization of $\mathbf{A}$. We implemented an analogous SYMAMD routine that is based on COLAMD.

Our test set included 50 matrices requiring 10^7 or more operations to factorize. The largest was a computational fluid dynamics problem from Ed Rothberg requiring 136 billion operations to factorize (CFD2). Our sample test matrices[5] are shown in Table VII. Results from SYMAMD, SYMMMD, AMDBAR, and MC47BD are shown in Tables VIII and IX. The time to construct $\mathbf{M}$ is included in the ordering time for SYMAMD and SYMMMD.

For these 9 matrices, SYMAMD was over six times faster than SYMMMD, on

[5]Every 7th matrix, in order of increasing ratio of SYMAMD to SYMMMD ordering quality, was chosen.

Table IX. Ordering quality

matrix	Nonzeros in $\mathbf{L}$ (10^3)				Cholesky flop. count (10^6)			
	SYM-AMD	SYM-MMD	AMD-BAR	MC47	SYM-AMD	SYM-MMD	AMD-BAR	MC47
PWT	1497	1425	1592	1556	156	140	173	162
MSC00726	103	108	111	111	20	22	23	23
BCSSTK38	735	803	752	737	118	143	127	119
NASA2146	135	157	140	140	11	15	12	12
CFD2	74832	94444	75008	75008	137470	211328	136476	136476
3DTUBE	26128	33213	26355	26355	29969	45578	30053	30053
GEARBOX	49268	63940	48556	48555	49849	88616	47068	47067
CRYSTM02	2025	3159	2286	2286	546	1240	719	719
FINAN512	2028	8223	2851	2838	249	12356	644	633
Median result relative to COLAMD	-	1.26	1.03	1.03	-	1.52	1.08	1.04

average. It nearly always produced significantly better orderings. In contrast, SYMAMD was always slower than AMDBAR and MC47BD, although it found orderings of similar quality (the FINAN512 is a notable exception, but none of these methods finds as good an ordering as a tree dissection method [Berger et al. 1995]).

An ordering algorithm designed for symmetric matrices (AMDBAR or MC47BD) is thus superior to one based on a column ordering of $\mathbf{M}$. There may be at least one important exception, however. In some applications, a better matrix $\mathbf{M}$ is available. Consider an n-by-n finite element matrix constructed as the summation of e finite elements. Each finite element matrix is a dense symmetric submatrix. Suppose element k is nonzero for all entries a_{ij} for which both i and j are in the set $\mathcal{E}_k$. We can construct a e-by-n matrix $\mathbf{M}$, where row k has the pattern $\mathcal{E}_k$. Since $\mathbf{M}^\mathsf{T}\mathbf{M}$ has the same pattern as $\mathbf{A}$, we can compute a column ordering of $\mathbf{M}$ and use it for the Cholesky factorization of $\mathbf{A}$. COLAMD would be faster than when using an $\mathbf{M}$ constructed without the knowledge of the finite element structure. The space to perform the ordering and symbolic analysis is less as well, since $\mathbf{M}$ has fewer nonzeros than $\mathbf{A}$. Although many of the matrices in our test set arise from finite element problems, only a few small ones are available in unassembled form, as a collection of finite elements. Thus, we are not able to evaluate this strategy on large, realistic test matrices.

6. SUMMARY

Two new ordering routines, COLAMD and SYMAMD, have been presented. For square nonsymmetric matrices, COLAMD is much faster and provides better orderings than MATLAB's COLMMD routine. It is also faster than symmetric-based ordering methods (such as AMDBAR), and uses less storage. For rectangular matrices (such as those arising in least squares problems and interior point methods for linear programming), COLAMD is faster than COLMMD and AMDBAR and finds orderings of comparable quality. We presented a symmetric ordering method

SYMAMD based on COLAMD; although it produces orderings as good as a truly symmetric ordering algorithm (AMDBAR), it is slower than AMDBAR.

COLAMD and SYMAMD are written in ANSI/ISO C, with MATLAB-callable interfaces. Version 2.3 of the code is freely available from the following sources:

(1) University of Florida, `http://www.cise.ufl.edu/research/sparse`.

(2) Netlib, `http://www.netlib.org/linalg/colamd/`.

(3) The MathWorks, Inc., for user-contributed contributions to MATLAB, `http://www.mathworks.com`. COLAMD and SYMAMD are built-in functions in MATLAB Version 6.0. In MATLAB Version 6.5, COLAMD is the default ordering method for the backslash operator when $\mathbf{A}$ is sparse and unsymmetric, and SYMAMD is the default when $\mathbf{A}$ is sparse and symmetric positive definite.

(4) The collected algorithms of the ACM, as Algorithm 8xx, described in [Davis et al. 20xx].

REFERENCES

Amestoy, P. R., Davis, T. A., and Duff, I. S. 1996. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Applic. 17,* 4, 886–905.

Amestoy, P. R., Davis, T. A., and Duff, I. S. 20xx. Algorithm 8xx: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw. xx,* x, xxx–xx.

Ashcraft, C. C. 1995. Compressed graphs and the minimum degree algorithm. *SIAM J. Sci. Comput. 16*, 1404–1411.

Bai, Z., Day, D., Demmel, J., and Dongarra, J. 1996. Test matrix collection (non-Hermitian eigenvalue problems), Release 1. Tech. rep., University of Kentucky. September. Available at ftp://ftp.ms.uky.edu/pub/misc/bai/Collection.

Berger, A. J., Mulvey, J. M., Rothberg, E., and Vanderbei, R. J. 1995. Solving multistage stochastic programs using tree dissection. Tech. Rep. SOR-95-07, Dept. of Civil Eng. and Operations Research, Princeton Univ., Princeton, NJ. June.

Carolan, W. J., Hill, J. E., Kennington, J. L., Niemi, S., and Wichmann, S. J. 1990. An empirical evaluation of the korbx algorithms for military airlift applications. *Operations Research 38,* 2, 240–248.

Coleman, T. F., Edenbrandt, A., and Gilbert, J. R. 1986. Predicting fill for sparse orthogonal factorization. *J. Assoc. Comput. Mach. 33*, 517–532.

Davis, T. A. 2000. Univ. of Florida sparse matrix collection. http://www.cise.ufl.edu/research/sparse.

Davis, T. A. and Duff, I. S. 1997. An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J. Matrix Anal. Applic. 18,* 1, 140–158.

Davis, T. A. and Duff, I. S. 1999. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans. Math. Softw. 25,* 1, 1–19.

Davis, T. A., Gilbert, J. R., Larimore, S. I., and Ng, E. G. 20xx. Algorithm 8xx: COLAMD, a column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw. xx,* x, xxx–xxx.

Demmel, J. W., Eisenstat, S. C., Gilbert, J. R., Li, X. S., and Liu, J. W. H. 1999. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Applic. 20,* 3, 720–755.

Dongarra, J. J., Du Croz, J., Duff, I. S., and Hammarling, S. 1990. A set of level-3 basic linear algebra subprograms. *ACM Trans. Math. Softw. 16,* 1, 1–17.

Dongarra, J. J. and Grosse, E. 1987. Distribution of mathematical software via electronic mail. *Commun. ACM 30*, 403–407.

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

Duff, I. S., Grimes, R. G., and Lewis, J. G. 1989. Sparse matrix test problems. *ACM Trans. Math. Softw. 15*, 1–14.

Duff, I. S., Grimes, R. G., and Lewis, J. G. 1992. Users' guide for the Harwell-Boeing sparse matrix collection (Release 1). Tech. Rep. RAL-92-086, Rutherford Appleton Laboratory, Didcot, Oxon, England. Dec.

Duff, I. S. and Reid, J. K. 1983. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Softw. 9,* 3, 302–325.

Feldmann, P., Melville, R., and Long, D. 1996. Efficient frequency domain analysis of large nonlinear analog circuits. In *Proceedings of the IEEE Custom Integrated Circuits Conference*. IEEE, Santa Clara, CA.

George, A. and Liu, J. W. H. 1980. An optimal algorithm for symbolic factorization of symmetric matrices. *SIAM J. Comput. 9*, 583–593.

George, A. and Liu, J. W. H. 1981. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, New Jersey.

George, A. and McIntyre, D. R. 1978. On the application of the minimum degree algorithm to finite element systems. *SIAM J. Numer. Anal. 15*, 90–111.

George, A. and Ng, E. 1985. An implementation of Gaussian elimination with partial pivoting for sparse systems. *SIAM J. Sci. Statist. Comput. 6,* 2, 390–409.

George, A. and Ng, E. 1987. Symbolic factorization for sparse Gaussian elimination with partial pivoting. *SIAM J. Sci. Statist. Comput. 8,* 6, 877–898.

Gilbert, J. R., Moler, C., and Schreiber, R. 1992. Sparse matrices in MATLAB: design and implementation. *SIAM J. Matrix Anal. Applic. 13,* 1, 333–356.

Gilbert, J. R. and Ng, E. G. 1993. Predicting structure in nonsymmetric sparse matrix factorizations. In *Graph Theory and Sparse Matrix Computation*, A. George, J. R. Gilbert, and J. W. H. Liu, Eds. Volume 56 of the IMA Volumes in Mathematics and its Applications. Springer-Verlag, New York, 107–139.

Gilbert, J. R., Ng, E. G., and Peyton, B. W. 1994. An efficient algorithm to compute row and column counts for sparse Cholesky factorization. *SIAM J. Matrix Anal. Applic. 15,* 4, 1075–1091.

Gilbert, J. R. and Peierls, T. 1988. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sci. Statist. Comput. 9*, 862–874.

Gustavson, F. G. 1978. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Trans. Math. Softw. 4,* 3, 250–269.

Hare, D. R., Johnson, C. R., Olesky, D. D., and Van Den Driessche, P. 1993. Sparsity analysis of the QR factorization. *SIAM J. Matrix Anal. Applic. 14,* 3, 665–669.

Heggernes, P. and Matstoms, P. 1994. Finding good column orderings for sparse QR factorization. Tech. rep., Dept. of Informatics, Univ. of Bergen, Bergen, Norway.

Karmarkar, N. 1978. A new polynomial time algorithm for linear programming. *Combinatorica 4*, 373–395.

Kern, J. L. 1999. Approximate deficiency for ordering the columns of a matrix. Tech. rep., Univ. of Florida, Gainesville, FL. (senior thesis, see http://www.cise.ufl.edu/colamd/kern.ps).

Larimore, S. I. 1998. An approximate minimum degree column ordering algorithm. Tech. Rep. TR-98-016, Univ. of Florida, Gainesville, FL. (Master's thesis, see http://www.cise.ufl.edu/tech-reports/).

Liu, J. W. H. 1985. Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Softw. 11,* 2, 141–153.

Liu, J. W. H. 1991. A generalized envelope method for sparse factorization by rows. *ACM Trans. Math. Softw. 17,* 1, 112–129.

Markowitz, H. M. 1957. The elimination form of the inverse and its application to linear programming. *Management Science 3*, 255–269.

Miller, J. J. H. and S., W. 1991. An exponentially fitted finite element method for a stationary convection-diffusion problem. In *Computatioal methods for boundary and interior layers in several dimensions*, J. J. H. Miller, Ed. Boole Press, Dublin, 120–137.

Ng, E. G. and Raghavan, P. 1999. Performance of greedy ordering heuristics for sparse Cholesky factorization. *SIAM J. Matrix Anal. Applic. 20,* 4, 902–914.

Resende, M. G. C., Ramakrishnan, K. G., and Drezner, Z. 1995. Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming. *Operations Research 43*, 781–791.

Rothberg, E. and Eisenstat, S. C. 1998. Node selection strategies for bottom-up sparse matrix orderings. *SIAM J. Matrix Anal. Applic. 19,* 3, 682–695.

Sherman, A. H. 1975. On the efficient solution of sparse systems of linear and nonlinear equations. Tech. Rep. 46, Yale Univ. Dept. of Computer Science. Dec.

Wright, S. J. 1996. *Primal-dual interior-point methods*. SIAM Publications, Philadelphia.

Yannakakis, M. 1981. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth. 2*, 77–79.

Zitney, S. E. 1992. Sparse matrix methods for chemical process separation calculations on supercomputers. In *Proc. Supercomputing '92*. IEEE Computer Society Press, Minneapolis, MN, 414–423.

Zitney, S. E., Mallya, J., Davis, T. A., and Stadtherr, M. A. 1996. Multifrontal vs. frontal techniques for chemical process simulation on supercomputers. *Comput. Chem. Eng. 20,* 6/7, 641–646.