ON THE COMPLEXITY OF SPARSE GAUSSIAN ELIMINATION VIA BORDERING [*]

RANDOLPH E. BANK[†] AND DONALD J. ROSE[‡]

Abstract. The complexity of a general sparse Gaussian elimination algorithm based on the bordering algorithm is analyzed. It has been shown that this procedure requires less integer overhead storage than more traditional general sparse procedures, but the complexity of the nonnumerical overhead calculations was not clear. Here the nonnumerical complexity of the original procedure is shown to be comparable to the numerical complexity for an $n \times n$ grid graph, and an enhancement of the procedure that can reduce the overhead is presented.

Key words. sparse Gaussian elimination, bordering, m-tree

AMS subject classifications. 65F05, 65N20

1. Introduction.

In this paper, we consider the solution of the $N \times N$ linear system

$$Ax = b \tag{1.1}$$

where A is large, sparse, symmetric, and positive definite. We consider the direct solution of (1.1) by means of general sparse Gaussian elimination. In such a procedure, we find a permutation matrix P, and compute the decomposition

$$PAP^t = LDL^t$$

where L is unit lower triangular and D is diagonal. The system (1.1) is then solved by

$$\begin{aligned} Lw &= Pb, \\ Dy &= w, \\ L^t z &= y, \\ x &= P^t z. \end{aligned}$$

Several good ordering algorithms (nested dissection and minimum degree) are available for computing P [5], [9]. Since our interest here does not focus directly on the ordering, we assume for convenience that $P = I$, or that A has been preordered to reflect an appropriate choice of P.

Our purpose here is to examine the nonnumerical complexity of the sparse elimination algorithm given in [3]. As was shown there, a general sparse elimination scheme based on the bordering algorithm requires less storage for pointers and row/column indices than more traditional implementations of general sparse elimination. This is accomplished by exploiting the m-tree, a particular spanning tree for the graph of the filled-in matrix. To our knowledge, the m-tree previously has not been applied in this

[*]Received by the editors September 23, 1987; accepted for publication November 15, 1988.

[†]Department of Mathematics, University of California at San Diego, La Jolla, California 92093. The work of this author was supported by the Office of Naval Research under contract N00014-82K-0197.

[‡]Department of Computer Science, Duke University , Durham, North Carolina 27706. The work of this author was supported by the Office of Naval Research under contract N00014-85K-0487, AT&T Bell Laboratories, and the Microelectronics Center of North Carolina.

fashion to the numerical factorization, but it has been used, directly or indirectly, in several optimal order algorithms for computing the fill-in during the symbolic factorization phase [4] - [8], [10], [12].

In §2, we review the bordering algorithm, and introduce the sorting and intersection problems that arise in the sparse formulation of the algorithm. In §3, we introduce m-trees (or elimination trees) and review their role in sparse Gaussian elimination. We do not attempt to present an overview here, but rather attempt to focus on those results that are relevant to our particular algorithm. This section assumes prior knowledge of the role of graph theory in sparse Gaussian elimination; surveys of this role are available in [9] and [5]. More general discussions of elimination trees are given in [6] - [8], [12].

In §4, we return to the sorting and intersection problems, and show how m-trees can be exploited effectively in their solution. The sorting problem is relatively straightforward, and its computational complexity is of lower order than the complexity of the numerical factorization. On the other hand, the complexity of the intersection problem is potentially of the same order as the numerical factorization; indeed, in our first formulation of the problem, it becomes clear that the complexity must be at least as great as the numerical factorization. Later we split the intersection problem into two parts, with one corresponding exactly to the numerical factorization, and the second being pure overhead. We then present a new procedure for reducing the complexity of this second part of the intersection problem; this procedure again exploits the structure of the m-tree.

In §5, we analyze the complexity of the old and new approaches to the intersection problem for the special case of an $n \times n$ grid ordered by nested dissection. The special structure of this problem allows us to make exact estimates of the complexity. For the old approach, we show that the complexity of the intersection problem is $O(n^3)$, the same as the complexity of the numerical computations [5], [11]. For the new approach, the complexity of the second part is reduced to $O(n^2(\log n)^2)$. In §6, we touch briefly on the issues of data structures and implementation.

We emphasize that in terms of a practical computer code for doing sparse Gaussian elimination, the best we realistically can expect to achieve for a package based on bordering is an execution time comparable to the better row-oriented general sparse matrix packages currently available (e.g., Yale Sparse Matrix Package [4] and Sparspak [5]), at least for sequential computation. Certainly the number of floating point computations in a general sparse code depends only on the ordering of the equations and the zero-nonzero structure of the original matrix, and this is the same for all procedures. The differences between algorithms are mainly in the ordering of the computations, data structures, and nonnumerical overhead. Here the bordering approach can offer some advantages. It usually requires less integer overhead storage [3] than row schemes, and since the storage required is not a function of the fill-in, the amount of integer overhead is known before the computation begins. Also, some sparse matrix problems present themselves in a way such that a columnwise sparse storage scheme coupled with the bordering algorithm for Gaussian elimination becomes the most convenient and obvious approach to their solution. Indeed, one such application (to the linear systems arising in the hierarchical basis multigrid method [1], [2]) motivated our original exploration of such algorithms.

In terms of nonnumerical computations, our new appoach to the intersection problem reduces nonnumerical computations in the numerical factorization phase to a level approximately equal to that of row-oriented schemes, that is, about one indirect address for each floating point multiplication operation in the inner loop in the symmetric case. For nonsymmetric problems with symmetric zero-nonzero patterns, the nonnumerical costs of our bordering approach remain the same, but the number of floating point operations approximately doubles. The number of nonnumerical computations in the forward/backward solution phases has always been about the same for the row- and column-oriented schemes. Thus we need not sacrifice execution time if it seems desirable to use a column-oriented approach.

2. The bordering algorithm and sparse elimination. Let A be a symmetric, positive definite matrix. We consider the factorization

$$A = LDL^t \tag{2.1}$$

where D is diagonal and L is unit lower triangular. Let A_k denote the $k \times k$ upper left principal submatrix of A, and we assume that we have already computed

$$A_{k-1} = L_{k-1} D_{k-1} L_{k-1}^t$$

by the bordering algorithm. Then

$$\begin{aligned} A_k &= \begin{bmatrix} A_{k-1} & c \\ c^t & \alpha \end{bmatrix} \\ &= L_k D_k L_k^t \\ &= \begin{bmatrix} L_{k-1} & 0 \\ \ell^t & 1 \end{bmatrix} \begin{bmatrix} D_{k-1} & 0 \\ 0 & \delta \end{bmatrix} \begin{bmatrix} L_{k-1}^t & \ell \\ 0 & 1 \end{bmatrix} \end{aligned}$$

where

$$\begin{aligned} L_{k-1} D_{k-1} \ell &= c, \\ \delta &= \alpha - \ell^t D_{k-1} \ell. \end{aligned}$$

Thus, at the kth stage, the bordering algorithm consists of solving the lower triangular system

$$L_{k-1} v = c \tag{2.2}$$

and setting

$$\ell = D_{k-1}^{-1} v, \tag{2.3}$$

$$\delta = \alpha - \ell^t v. \tag{2.4}$$

Elementwise, the algorithm may be written in the following manner.

Procedure Dense Factor.

```
(D1)    for $k = 1, N$
(D2)        $d_{kk} \leftarrow a_{kk}$
(D3)        for $j = 1, k-1$
(D4)            $v_j = a_{jk} - \sum_{i=1}^{j-1} \ell_{ji} v_i$
(D5)            $\ell_{kj} = v_j / d_{jj}$
(D6)            $d_{kk} \leftarrow d_{kk} - \ell_{kj} v_j$
```

Let $\mathcal{C}_k$ denote the index set of nonzeroes in column k of $L^t - I$ (row k of $L - I$). Then, for sparse matrices, the factorization algorithm may be written as follows:

Procedure Sparse Factor.

```
(S1)    for $k = 1, N$
(S2)        $d_{kk} \leftarrow a_{kk}$
(S3)        for $j \in \mathcal{C}_k$
(S4)            $v_j = a_{jk} - \sum_{i \in \mathcal{C}_k \cap \mathcal{C}_j} \ell_{ji} v_i$
(S5)            $\ell_{kj} = v_j / d_{jj}$
(S6)            $d_{kk} \leftarrow d_{kk} - \ell_{kj} v_j$
```

Because of the implicit nature of line (S4), the indices j on line (S3) must be sorted such that the right-hand side of line (S4) is always well defined. Sorting $\mathcal{C}_k$ by increasing order is certainly sufficient, but other orderings are possible and will prove more convenient. In particular, any sorting of the indices that allows (2.2) to be backsolved is acceptable. We will refer to this as the *sorting problem.*

The computation of v_j in line (S4) requires the computation of $\mathcal{C}_k \cap \mathcal{C}_j$. We will refer to this as the *intersection problem.* Since this is an inner loop computation, it clearly contributes to the highest order term of the overall complexity; thus it is important to compute these intersections as efficiently as possible. We will analyze these problems in §§4 and 5.

3. m-trees and sparse elimination. Let $\mathcal{G} = (\mathcal{X}, \mathcal{E}, \alpha)$ be the connected, ordered graph associated with the irreducible, symmetric, positive definite matrix $A = LDL^t$. Here $\mathcal{X} = \{x_i\}_{i=1}^N$ denotes the vertex set, $\mathcal{E}$ the edge set ($e_{ji} = e_{ij} \in \mathcal{E}, i \neq j$ if and only if $a_{ij} \neq 0$), and α is the ordering ($\alpha(i) = x_i$). Let $\mathcal{G}' = (\mathcal{X}, \mathcal{E} \cup \mathcal{F}, \alpha)$ denote the chordal graph generated by α. $\mathcal{F}$ denotes the set of fill-in edges generated by the elimination process.

For $x_i \in \mathcal{X}$,

$$\text{adj}(x_i) = \{y_j | e_{ij} \in \mathcal{E} \cup \mathcal{F}\}$$

denotes the adjacency of x_i in $\mathcal{G}'$. We denote the monotone adjacency of x_i by

$$\text{madj}(x_i) = \{y_j \in \text{adj}(x_i) | j > i\}$$

and set

$$\text{cadj}(x_i) = \text{adj}(x_i) - \text{madj}(x_i).$$

The index set associated with madj(x_i) is the set of column indices for row i of $L^t - I$, while cadj(x_i) corresponds to $\mathcal{C}_i$. Additionally, recall that madj(x_i) is a clique in $\mathcal{G}'$.

Let $\mathcal{G}' = (\mathcal{X}, \mathcal{E} \cup \mathcal{F}, \alpha)$ be a chordal graph and let $m(i), 1 \leq i \leq N - 1$ be given by

$$m(i) = \min\{j | x_j \in \text{madj}(x_i)\}. \tag{3.1}$$

Then the *m-tree* $\mathcal{T}$ for $\mathcal{G}'$ is the tree with vertex set $\mathcal{X}$ and edges $\mathcal{E}' = \{e_{im(i)}\}_{i=1}^{N-1}$. The m-tree is also called the *elimination tree* by Liu [7] and Schreiber [12]. See Liu [8] for a recent survey of the role of m-trees in sparse Gaussian elimination. Among

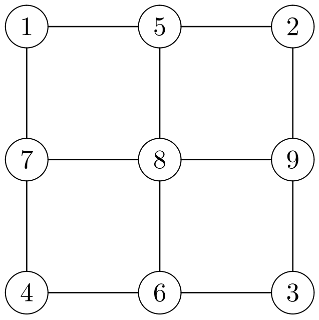

FIG. 3.1. *A* 3×3 *grid graph with nested dissection ordering.*

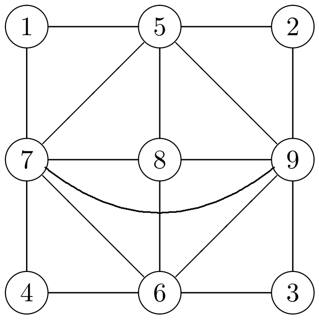

FIG. 3.2. *Fill-in for the* 3×3 *grid graph.*

the m-tree's more important applications is its use in optimal order procedures for computing the fill-in $\mathcal{E} \cup \mathcal{F}$ using the ideas of Rose, Tarjan, and Lueker [10].

As an example, consider the 3×3 grid graph with the nested dissection ordering α, illustrated in Figs. 1 and 2.

For this graph $m(i)$ is defined as follows:

i	1	2	3	4	5	6	7	8	9
$m(i)$	5	5	6	6	7	7	8	9	–

The m-tree is shown in Fig. 3.

LEMMA 3.1. *Let $\mathcal{G}' = (\mathcal{X}, \mathcal{E} \cup \mathcal{F}, \alpha)$ be chordal and let $e_{ij} \in \mathcal{E} \cup \mathcal{F}, i < j$. Then either $m(i) = j$ or $e_{m(i)j} \in \mathcal{E} \cup \mathcal{F}$.*

Proof. See Schreiber [12]. We give proofs of this and other lemmas in this section because of their brevity. If $m(i) = j$, we are done, so we assume that $m(i) = k < j$. Then, since $\mathrm{madj}(x_i)$ is a clique and $x_j, x_k \in \mathrm{madj}(x_i)$, $e_{kj} \in \mathcal{E} \cup \mathcal{F}$. ☐

LEMMA 3.2. *Let $\mathcal{T}_i$ be the subgraph of $\mathcal{T}$ induced by the set $\{x_i\} \cup cadj(x_i)$. Then $\mathcal{T}_i$ is connected (i.e., it is a subtree).*

Proof. See Schreiber [12]. We will show the path from $x_j \in \mathrm{cadj}(x_i)$ to x_i in $\mathcal{T}$ contains only vertices in the set $\{x_i\} \cup \mathrm{cadj}(x_i)$. This is done by induction on ℓ, the length of the path. If $\ell = 1$, then $m(j) = i$, and $e_{ij} \in \mathcal{E}'$. We assume the lemma is true for paths of length $\ell - 1$, and show it for a path of length ℓ. Let the path from $x_j \in \mathrm{cadj}(x_i)$ to x_i in $\mathcal{T}$ be of length ℓ, and let $k = m(j) < i$. Note that $e_{ki} \in \mathcal{E} \cup \mathcal{F}$.

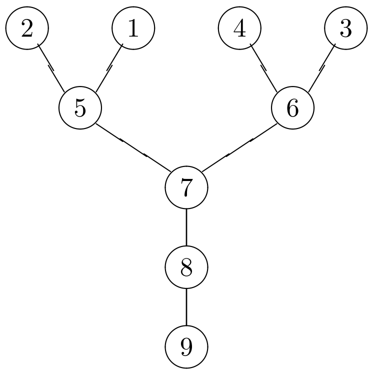

FIG. 3.3. *An m-tree for the* 3×3 *grid graph.*

Thus, $x_k \in \text{cadj}(x_i)$ and the length of the path from x_k to x_i in $\mathcal{T}$ is $\ell - 1$. □ The subtrees for the example 3×3 grid graph are shown in Fig. 4.

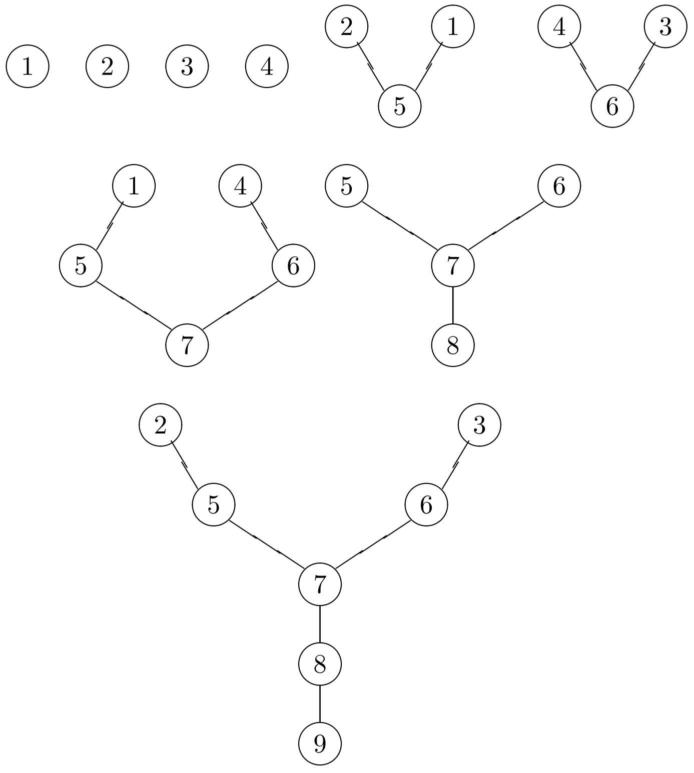

FIG. 3.4. *Subtrees for the* 3×3 *grid graph.*

Let $\mathcal{L}_i = \{x_j \in \text{cadj}(x_i) | x_j \text{ is a leaf of } \mathcal{T}_i\}$ denote the set of leaves of $\mathcal{T}_i$. For $x_j \in \mathcal{L}_i$, $m(j) \neq i$, let $\{x_{\ell_1}, x_{\ell_2}, \cdots, x_{\ell_k}\}$ be the path of length $k-1 \geq 2$ from $x_j = x_{\ell_1}$ to $x_i = x_{\ell_k}$ in $\mathcal{T}$. The edge e_{ij} is called a *backedge.* Since $x_{\ell_p} \in \text{cadj}(x_i), 2 \leq p \leq k-1$,

$e_{\ell_p i} \in \mathcal{E} \cup \mathcal{F}$. Thus the path and the backedge form a cycle in $\mathcal{G}'$; this cycle is chorded by the edges $e_{\ell_p i}, 2 \leq p \leq k-1$. Let $\mathcal{B}$ denote the set of backedges and $\mathcal{K}$ the set of chords. It is easy to see

$$\mathcal{E} \cup \mathcal{F} = \mathcal{E}' \cup \mathcal{B} \cup \mathcal{K}.$$

Following Liu [7], the graph $\mathcal{S} = (\mathcal{X}, \mathcal{E}' \cup \mathcal{B})$ is called the *skeleton* of $\mathcal{G}'$. The skeleton of the 3×3 grid graph is shown in Fig. 5.

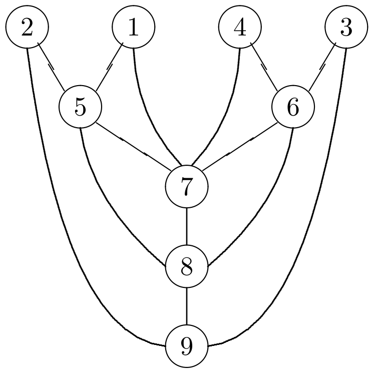

FIG. 3.5. *Skeleton for the* 3×3 *grid graph.*

LEMMA 3.3. *Let $\mathcal{G}$, $\mathcal{G}'$, $\mathcal{E} \cup \mathcal{F}$, and $\mathcal{B}$ be defined as above. Then $\mathcal{B} \subset \mathcal{E}$.*

Proof. See Liu [7] and Schreiber [12]. Let $e_{ij} \in \mathcal{B}, j < i$. Then $x_j \in \mathcal{L}_i$. Suppose $e_{ij} \notin \mathcal{E}$, so $e_{ij} \in \mathcal{F}$. Define

$$k = \max\{p | x_i, x_j \in \text{madj}(x_p)\}.$$

Clearly k exists; otherwise, e_{ij} would not be in $\mathcal{F}$. We now observe that $m(k) = j$; if $\ell = m(k) < j$, then $x_\ell, x_i, x_j \in \text{madj}(x_k)$. Since $\text{madj}(x_k)$ is a clique, $x_i, x_j \in \text{madj}(x_\ell)$, contradicting the definition of k. However, since $m(k) = j$, $e_{kj} \in \mathcal{E}'$, contradicting $x_j \in \mathcal{L}_i$. □

Since $\mathcal{B} \subseteq \mathcal{E}$, the index sets corresponding to the $\mathcal{L}_i$ are subsets of the row indices for the upper triangular part of column i of the matrix A.

4. The sorting and intersection problems.

It is apparent that the generation of the sets $\mathcal{C}_k$ required for the numerical factorization requires only knowledge of the sets $\mathcal{L}_k \subseteq \mathcal{C}_k$ and the m-tree for $\mathcal{G}'$. The m-tree $\mathcal{T}$, along with the sets $\mathcal{C}_k$ for all k, can be computed in $O(|\mathcal{E} \cup \mathcal{F}|)$ time using the procedures in [3]. The $\mathcal{C}_k$ need not be permanently stored, since they can be regenerated as needed using the $\mathcal{L}_k$ and the m-tree.

By Lemma 3.3, the index set for $\mathcal{L}_k$ is a subset of the row indices for the nonzeros in the strict upper triangular part of column k of A. It is thus convenient to store the strict upper triangular part of A column by column, since this also facilitates the use of the bordering algorithm. It is not essential that the index set $\mathcal{L}_k$ be explicitly determined; indeed, it is convenient to define *generalized leaves* $\mathcal{L}'_k$ by

$$\mathcal{L}'_k = \{x_j \in \mathcal{C}_k | e_{kj} \in \mathcal{E}, j < k\}.$$

The index set for $\mathcal{L}'_k$ corresponds exactly to the row indices for the nonzeros in the strict upper triangle of column k of A; clearly $\mathcal{L}_k \subseteq \mathcal{L}'_k$.

We next partition the index set $\mathcal{C}_i$ among the generalized leaves for $\mathcal{C}_i$. Thus we let

$$\mathcal{C}_i = \bigcup_{x_j \in \mathcal{L}'_i} \mathcal{D}_{ij},$$
$$\mathcal{D}_{ij} \cap \mathcal{D}_{ik} = \emptyset, \quad j \neq k.$$

The sets $\mathcal{D}_{ij}$ are defined as follows: for $x_j \in \mathcal{L}'_i$, $\mathcal{D}_{ij}$ is the index set of vertices on the path from x_j to x_i in $\mathcal{T}$ which do not coincide with the path of any higher ordered vertex in $\mathcal{L}'_i$. For example, in our 3×3 grid graph, we have for x_9,

$$\begin{aligned} \mathcal{L}_i &= \{x_3, x_2\}, \\ \mathcal{L}'_i &= \{x_8, x_3, x_2\}, \\ \mathcal{D}_{98} &= \{8\}, \\ \mathcal{D}_{93} &= \{3, 6, 7\}, \\ \mathcal{D}_{92} &= \{2, 5\}. \end{aligned}$$

This partitioning of $\mathcal{C}_i$ results naturally if the vertices in $\mathcal{L}'_i$ are sorted by index and $\mathcal{C}_i$ is generated by processing $x_j \in \mathcal{L}'_i$ in *decreasing* order. Since $|\mathcal{L}'_i|$ is typically not large, sorting these sets as an initialization step generally does not contribute to the highest order complexity terms. Thus we assume that the sorted $\mathcal{L}'_i$ are available as input.

For any $x_j \in \mathcal{L}'_i$, the set $\mathcal{D}_{ij}$ is generated from j and $\ell = |\mathcal{D}_{ij}|$ using the m-function,

$$\mathcal{D}_{ij} = \{j, m(j), m(m(j)), \cdots, m^{\ell-1}(j)\}. \tag{4.1}$$

We now return to the two problems mentioned at the conclusion of §2. We consider first the sorting problem for $\mathcal{C}_k$ in line (S3) of Procedure Sparse Factor. In light of the analysis of §3, we must sort the vertices in cadj(x_k), which together with x_k are the vertices of $\mathcal{T}_k$, such that the predecessors of vertex $x_j \in$ cadj(x_k) are ordered before x_j itself. If this is done, the right-hand side of line (S4) of Procedure Sparse Factor always will be well defined. One such sorting can be generated easily by processing the vertices in $\mathcal{L}'_k$ in *increasing* order, generating the sets $\mathcal{D}_{kj}$ using the m-function. For example, for vertex x_9 of our 3×3 grid graph, this would result in the ordering

$$\begin{aligned} \mathcal{C}_9 &= \{2, 5, 3, 6, 7, 8\} \\ &= \mathcal{D}_{92} \cup \mathcal{D}_{93} \cup \mathcal{D}_{98}. \end{aligned}$$

Another solution to the sorting problem, based on a renumbering of the vertices using a postorder traversal of the m-tree, is given by Liu [7].

As each element $j \in \mathcal{C}_k$ is generated, we can mark an integer vector c, initialized to zero, to mark the set $\mathcal{C}_k$. It is thus easy to test if $i \in \mathcal{C}_k$ for any i by checking if $c(i) \neq 0$. We assume the existence of such an array as we analyze the intersection problem.

Given $\mathcal{C}_k$, represented by the array c, the problem of computing $\mathcal{C}_i \cap \mathcal{C}_k$ can be done in $O(|\mathcal{C}_i|)$ time by generating

$$\mathcal{C}_i = \bigcup_{j \in \mathcal{L}'_i} \mathcal{D}_{ij}$$

and testing using c. Let $\{\ell_p\}_{p=1}^{|\mathcal{D}_{ij}|}$ be the ordered sequence of indices in $\mathcal{D}_{ij}$. Then, assuming $\mathcal{D}_{ij} \cap \mathcal{C}_k \neq \emptyset$, there will exist a $\overline{p}$ such that

$$\ell_p \notin \mathcal{C}_k, \qquad 1 \leq p \leq \overline{p} - 1, \tag{4.2}$$

$$\ell_p \in \mathcal{C}_k, \qquad \overline{p} \leq p \leq |\mathcal{D}_{ij}|. \tag{4.3}$$

This is true since $\mathcal{T}_k$ is a connected subtree of $\mathcal{T}$ and the sequence $\{\ell_p\}$ corresponds to a path in $\mathcal{T}$ generated using the m-function.

Clearly, the second part of the sequence (4.3) generates actual floating point computations on line (S4) of Procedure Sparse Factor, and thus the complexity of this portion is unavoidably of the same order as the floating point work. On the other hand, generating the first part of the sequence is nonnumerical overhead which does not correspond to anything useful in terms of the numerical factorization.

Since the computation of intersections must contribute to the highest order complexity term, we are interested in finding a procedure for reducing the wasted computuation in (4.2). We are thus led to define, for each $\mathcal{D}_{ij}$, $i \in \mathcal{C}_k$, the index

$$q_{ijk} = \left\{ \begin{array}{ll} \ell_{\overline{p}} & \mathcal{D}_{ij} \cap \mathcal{C}_k \neq \emptyset \\ 0 & \mathcal{D}_{ij} \cap \mathcal{C}_k = \emptyset \end{array} \right\} \tag{4.4}$$

for $j < i < k$. We then recast the intersection problem in terms of computing q_{ijk} as quickly as possible, and then view the complexity of the intersection problem in terms of the complexity of computing q_{ijk}.

Let $s \leq t < r$ be integers; we now define the *run* $R = R(r,s) = \{t\}_{t=s}^{r-1}$ such that

$$\begin{gathered} m(t) = t+1 \qquad s \leq t \leq r-1 \\ m(r) \neq r+1 \qquad \text{if } r \neq N \\ m(s-1) \neq s \qquad \text{if } s \neq 1 \end{gathered} \tag{4.5}$$

Note that a run must contain at least one vertex (in which case $R(r,s) = \{s\}$, $m(s) = s+1$).

It is well known that sparse Gaussian elimination tends to produce large cliques in $\mathcal{G}'$; most of the vertices in these cliques will have $m(k) = k+1$. On the other hand, the formation of a large clique is not a necessary condition for the formation of a large run; for example, $m(k) = k+1$ for all $k < N$ in a tridiagonal matrix.

In any event, because of the restrictions on $m(r)$ and $m(s-1)$ in (4.5), a given integer t can be in at most one run. Thus we are able to define an *express vector* $e(t)$ which allows us to examine all the vertices in a run in $O(1)$ work. The express vector e, of length N is given by

$$e(t) \left\{ \begin{array}{ll} = r & \text{if } t \in R(r,s) \text{ for some } r \text{ and } s\ , \\ \leq 0 & \text{otherwise.} \end{array} \right.$$

Initially, we set $e(k) = 0$ if k is not in a run. In particular, note that if r is at the end of the run $R(r,s)$ then $e(r) = 0$. The use of the express vector results in a particular type of path compression in the m-tree, which maintains the structure that is crucial in the solution of the intersection problem.

Suppose a run $R(r,s) \cap \mathcal{C}_k \neq \emptyset$. We then (temporarily) set $e(r) = -\min\{\ell \in R(r,s) \cap \mathcal{C}_k\}$; that is, $-e(r)$ points to the lowest numbered vertex in the run that is also in $\mathcal{C}_k$. This can be determined easily as the marker array c is being computed (for each $i \in \mathcal{C}_k$, check if $e(i) > 0$). Given the arrays e, c, and m, and the integer $|\mathcal{D}_{ij}|$, the following procedure computes q_{ijk}.

Procedure Get_q_{ijk}.

```
(G1)     $q \leftarrow j$; $count \leftarrow 1$
(G2)     while $c(q) = 0$ and $count \leq |\mathcal{D}_{ij}|$ do
(G3)         if $e(q) \leq 0$ then
(G4)             $q \leftarrow m(q)$
(G5)             $count \leftarrow count + 1$
(G6)         else
(G7)             $q' \leftarrow q$
(G8)             $q \leftarrow e(q)$
(G9)             if $e(q) < 0$ then $q \leftarrow -e(q)$
(G10)            $count \leftarrow count + q - q'$
(G11)        end if
(G12)    end while
(G13)    if $c(q) = 0$ or $count > |\mathcal{D}_{ij}|$ , then $q \leftarrow 0$
```

This procedure simply generates the sequence (4.1) for $\mathcal{D}_{ij}$, looking for q_{ijk}. When it is possible to do so, we use the express vector to process runs in $O(1)$ work.

5. The intersection problem for grid graphs. Let $\mathcal{G}(k)$ be the $n \times n$ grid graph with $n = 2^k - 1$, ordered using nested dissection ($N = n^2$). $\mathcal{G}'(k)$ and $\mathcal{T}(k)$ will denote the triangulation of $\mathcal{G}(k)$ and its m-tree, respectively. Recall that nested dissection orders the vertices in a cross-shaped separater $\mathcal{S}$, consisting of the $2n-1$ vertices in row $(n+1)/2$ and column $(n+1)/2$ last. This leaves four grid graphs $\mathcal{G}'(k-1)$ to be recursively ordered. This is shown schematically in Fig. 6.

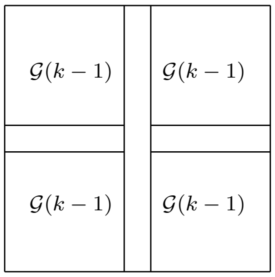

FIG. 5.1. *Nested dissection ordering.*

The recursive nature of the ordering is reflected in the m-tree which has the recursively defined structure shown in Fig. 7.

The nodes labeled $\mathcal{T}(k-1)$ are m-trees for the four subgraphs $\mathcal{G}'(k-1)$. Note that, with the exceptions of vertices x_{p+m} and x_n, $m(j) = j+1$ for all $x_j \in \mathcal{S}$.

LEMMA 5.1. *For each $x_i \in \mathcal{X}$ in $\mathcal{G}(k)$, $|\mathcal{L}_i| \leq 2$.*

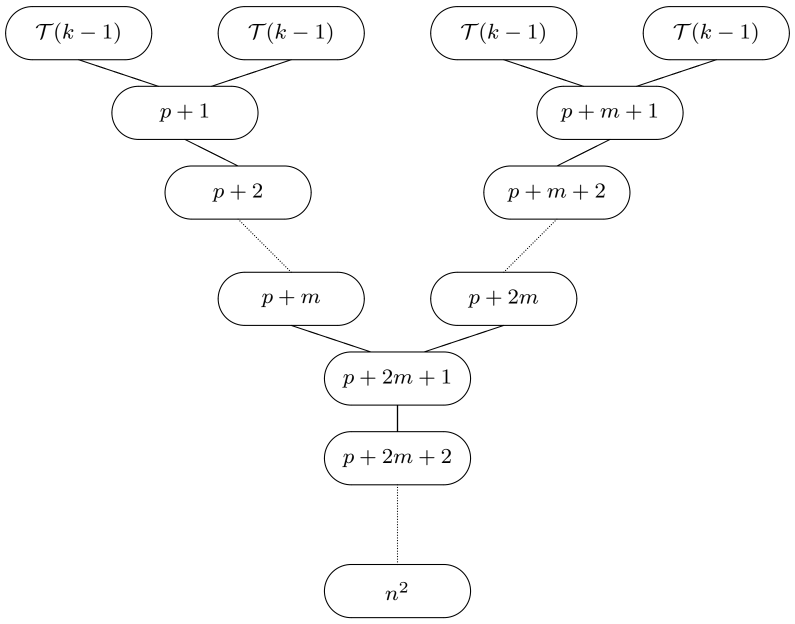

FIG. 5.2. *m-tree for the nested dissection ordering.* $p = n^2 - 2n + 2$, $m = (n-1)/2$ *and* $n = 2^k - 1$.

Proof. Since $|\mathcal{L}_i'| \leq 4$, there are at most four possible leaves for any vertex. The proof is by induction on k; the case $k = 1$ is trivial. Using the induction hypothesis for the four subgraphs $\mathcal{G}(k-1)$, we are left to consider only $x_i \in \mathcal{S}$. For such an x_i, at most two vertices in $\mathcal{L}_i'$ are not also in $\mathcal{S}$. By considering all the special cases, we straightfowardly surmise that $|\mathcal{L}_i| \leq 2$ for all $x_i \in \mathcal{S}$. □

LEMMA 5.2. *The height $h(k)$ of $\mathcal{T}(k)$ is*

$$h(k) = 3(2^k - 2) - 2(k-1) \tag{5.1}$$

Proof. Evidently

$$h(k) = h(k-1) + 3 * 2^{k-1} - 2.$$

with $h(1) = 0$. The solution of the difference equation is given in (5.1). □

Note also from (5.1) that

$$h(k) \leq 3n.$$

Thus we have the following lemma from Lemmas 5.1 and 5.2.

LEMMA 5.3. *For any $x_i \in \mathcal{G}'(k)$,*

$$|\mathcal{C}_i| \leq 6n.$$

THEOREM 5.4. *The complexity of the intersection problem without using the express function is $O(n^3)$.*

Proof. We estimate the cost $F(n)$ for computing q_{ijk} for all the relevant indices within the context of the numerical factorization. The procedure uses only the m-function, and thus must generate all entries in (4.2).

First, consider the cost for a single $x_i \in \mathcal{S}$. By Lemmas 5.1-5.3, the cost will be at most $O(n^2)$, since $|\mathcal{L}_j|$ and $|\mathcal{L}'_j|$ are bounded by constants, and $|\mathcal{C}_j| \leq O(n)$ for all $x_j \in \mathcal{X}$. Since $|\mathcal{S}| = 2n-1$, the cost for all vertices in $\mathcal{S}$ is $O(n^3)$. Thus,

$$F(n) \leq 4F(n/2) + \gamma n^3 \tag{5.2}$$

for some constant γ. The solution of the majorizing difference equation shows $F(n) \leq O(n^3)$. This is the same complexity as the numerical factorization. Thus, although the intersection problem contributes to the highest order complexity term, it does not increase the overall order of complexity. □

We now consider solving the intersection problem using the express vector and the m-function, as in Procedure Get_q_{ijk}. We let $Q(k)$ be the cost of Procedure Get_q_{ijk} for $\mathcal{T}(k)$. Then, for any j, the cost of processing the portion of the first sequence in (4.2) that lies in $\mathcal{S}$ is $O(1)$, since all but two vertices in $\mathcal{S}$ are in runs. Thus

$$Q(k) \leq Q(k-1) + \gamma$$

for some constant γ. The solution of this difference equation shows

$$Q(k) \leq \gamma k + O(1) = O(\log n).$$

Using Lemmas 5.1-5.3, we have

$$F(n) \leq 4F(n/2) + \gamma n^2 \log n$$

instead of (5.2). The solution of the majorizing difference equation shows

$$F(n) \leq O(n^2 (\log n)^2).$$

Thus we have shown the following theorem.

THEOREM 5.5. *The complexity of the intersection problem using the express function is* $O(n^2(\log n)^2)$.

In this case the intersection problem does not contribute to the highest order complexity terms.

In Table 1, we compare the actual nonnumerical overhead for the two procedures for $n \times n$ grid graphs with $n = 2^k - 1$, $3 \leq k \leq 8$, using the nested dissection ordering. The row labeled f lists the number of floating point operations used on line (S4) of Procedure Sparse Factor; this also counts the number of unavoidable indirect addresses corresponding to indices in the intersection $\mathcal{C}_i \cap \mathcal{C}_j$. The row labeled m counts the number of times line (G4) of Procedure Get_q_{ijk} is executed, while the row labeled e counts the number of times line (G8) is executed. Thus the sum $m + e$ reflects the nonnumerical overhead associated with solving the intersection problem using the express vector. Finally, the row labeled s gives the number of saved indirect addresses; $m + e + s$ reflects the nonnumerical overhead in solving the intersection problem without the express vector. This can be modeled using Procedure Get_q_{ijk} with $e(i) = 0$ for all i.

In Table 1, both f and s grow as $O(n^3)$ complexity; $m+e$ is growing as $O(n^2(\log n)^2)$. The behavior illustrated here seems to be typical of general grid problems arising from finite element or finite difference discretizations of partial differential equations. We

TABLE 5.1

Nonnumerical overhead for grid graphs.

k	3	4	5	6	7	8
N	49	225	961	3,969	16,129	65,025
f	580	11,496	153,668	1,664,596	15,963,924	142,335,428
m	448	6,684	66,392	524,564	3,585,936	22,215,844
e	72	2,256	35,460	332,716	2,519,844	16,693,084
s	66	3,814	95,094	1,510,526	18,581,430	195,752,462

have had the most experience with problems posed on irregular and nonuniform triangular meshes; there we have noticed that without the express vector, in large problems, $50-60$ percent of the indirect addresses (generated as $k \leftarrow m(k)$) used in solving the problem do not have corresponding floating point operations. On the other hand, with the express vector, the ratio of indirect addresses to floating point operations is close to one (or one half for nonsymmetric problems retaining a symmetric zero-nonzero structure). Thus, a general sparse matrix code based on the bordering algorithm should have execution times comparable to the current generation of general sparse matrix packages based on rowwise Cholesky factorization (e.g., Yale Sparse Matrix Package [4] and Sparspak [5]).

6. A note on data structures and implementation. Although it is possible to implement the sparse bordering algorithm using only $O(N)$ integer storage for the factored matrix (all temporary work space), we favor the data structure described in [3], which requires an integer array of length $NZ+1$, where NZ is the number of nonzeros in the upper triangle of A. This allows for a relatively simple and more time-efficient code. At the same time, $NZ = O(N)$ for many sparse matrix problems, for example, systems arising from discretizations of partial differential equations. We briefly summarize the data structures proposed in [3], restricted here to the case of symmetric matrices, and discuss the practical implementation of our procedures within this framework. [1]

Nonzeros in the upper triangle of the sparse matrix A are stored in an array a of length $NZ+1$; elements are referenced through an integer array ja, also of length $NZ+1$. Nonzeros in the Cholesky factor $L = U^t$ and the diagonal matrix D are stored in an array u of length $NZ'+1$, where NZ' is the number of nonzeros in $D+U$. Entries in u are referenced through ja and an integer array jl of length $NZ+1$, the same length as ja.

The entries of a and ja are defined as follows: $a(i), 1 \leq i \leq N$ contain the ith diagonal entry of A (a_{ii}). Entries $a(ja(i))$ to $a(ja(i+1)-1)$, $1 \leq i \leq N$ contain the nonzeros in the strict upper triangular part of the ith column of A, stored in order of decreasing row index; corresponding entries of the ja array contain the row indices. The entry $a(N+1)$ is arbitrary, and is included because $N+1$ pointers are required at the beginning of ja. This scheme, although different in detail, was motivated by the data structures of the Yale Sparse Matrix Package [4], except that the roles of rows and columns have been interchanged.

The array u is somewhat similar in structure to a. The first N locations of u contain the reciprocals of the diagonal entries of D; $u(N+1)$ is arbitrary. The

[1] A prototype package for carrying out general sparse Gaussian elimination using the bordering approach (essentially identical to that presented in [3] except for the inclusion of an express vector in the numerical factorization routine) is available from Argonne National Laboratory via Netlib.

TABLE 6.1
Data structure entries for 3×3 grid graph.

i	$ja(i)$	$a(i)$	$jl(i)$	$u(i)$
1	11	a_{11}	5	d_{11}^{-1}
2	11	a_{22}	5	d_{22}^{-1}
3	11	a_{33}	6	d_{33}^{-1}
4	11	a_{44}	6	d_{44}^{-1}
5	11	a_{55}	7	d_{55}^{-1}
6	13	a_{66}	7	d_{66}^{-1}
7	15	a_{77}	8	d_{77}^{-1}
8	17	a_{88}	9	d_{88}^{-1}
9	20	a_{99}		d_{99}^{-1}
10	23		11	
11	2	a_{25}	12	u_{25}
12	1	a_{15}	13	u_{15}
13	4	a_{46}	14	u_{46}
14	3	a_{36}	15	u_{36}
15	4	a_{47}	17	u_{47}
16	1	a_{17}	19	u_{67}
17	7	a_{78}	20	u_{17}
18	6	a_{68}	21	u_{57}
19	5	a_{58}	22	u_{78}
20	8	a_{89}	23	u_{68}
21	3	a_{39}	26	u_{58}
22	2	a_{29}	28	u_{89}
23				u_{39}
24				u_{69}
25				u_{79}
26				u_{29}
27				u_{59}

following entries in u contain the nonzeros in the strict upper triangular part of U, stored column by column; the entries are ordered corresponding to the sorting of the indices generated by our solution of the sorting problem. The diagonal entries of U are unity and are not stored.

The array jl is used in conjunction with ja to access the data in u. Recall that each row index stored in ja corresponds to one of the generalized leaves in the sets $\mathcal{L}_i'$; this is really the key observation in reducing the overhead storage. Entries $jl(i), 1 \leq i \leq N-1$ contain $m(i)$, yielding the $N-1$ edges in the m-tree. Entry $jl(N)$ is arbitrary. Entries $jl(i-1)$ (and $jl(i)-1$), $N+2 \leq i \leq NZ+1$ point at the first (and last) entries in u where the nonzeros of U generated by the leaf index $ja(i)$ are stored. These nonzeros correspond to one of the ordered index sets $\mathcal{D}_{jk}$ defined in this work. The row index for the first entry is of course given by $ja(i)$; subsequent indices are generated in the proper order using the m-tree.

The complete set of data structures as they appear for our simple 3×3 grid graph is shown in Table 2.

The following looping structure accesses the nonzeros in column k of U in sorted order:

```
(A1)        for $leaf = ja(k+1)-1, ja(k)$, step $-1$
(A2)            $j \leftarrow ja(leaf)$
(A3)            for $jloc = jl(leaf-1), jl(leaf)-1$, step 1
(A4)                ($u(jloc)$ contains matrix entry $u_{jk}$)
(A5)                $j \leftarrow jl(j)$
(A6)            end for
(A7)        end for
```

Statement (A1) loops over the generalized leaves for $\mathcal{T}_k$. The loop (A3)-(A6) generates the ordered index set $\mathcal{D}_{kj}$ associated with that leaf; the m-tree is used in (A5). By the definition of jl, the parameter $jloc$ always points to the entry in the Cholesky factor $u_{jk} = \ell_{kj}$.

Variations on the looping structure (A1)-(A7) suffice for the forward and backward solution procedures using this data structure; for these procedures, the leaves can be processed in either forward or reverse order. This also suffices for the middle loop, line (S3) in Procedure Sparse Factor. The inner loop on line (S4) is of this form, except that the starting value for the line corresponding to (A3) (i.e., $jl(leaf-1)$) is replaced by a value determined by Procedure Get_q_{ijk}. For this data structure, a simple modification of parameter *count* in Procedure Get_q_{ijk} will allow *count* to return the starting index for line (A3), a value $jl(leaf-1) \leq count \leq jl(leaf)-1$, provided that the intersection is not empty.

REFERENCES

[1] R. E. Bank, PLTMG *users' guide, edition 5.0*, tech. report, Department of Mathematics, University of California, San Diego, CA, 1988.

[2] R. E. Bank, T. F. Dupont, and H. Yserentant, *The hierarchical basis multigrid method*, Numer. Math., 52 (1988), pp. 427–458.

[3] R. E. Bank and R. K. Smith, *General sparse elimination requires no permanent integer storage*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 574–584.

[4] S. C. Eisenstat, M. C. Gursky, M. Schultz, and A. Sherman, *Algorithms and data structures for sparse symmetric gaussian elimination*, SIAM J. Sci. Stat. Comput., 2 (1982), pp. 225–237.

[5] A. George and J. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, Englewood Cliffs, NJ, 1981.

[6] K. H. Law and S. J. Fenves, *A node addition model for symbolic factorization*, ACM TOMS, 12 (1986), pp. 37–50.

[7] J. W. H. Liu, *A compact row storage scheme for cholesky factors using elimination trees*, ACM TOMS, 12 (1986), pp. 127–148.

[8] ———, *The role of elimination trees in sparse factorization*, Tech. Report CS-87-12, Department of Computer Science, York University, Ontario, Canada, 1987.

[9] D. J. Rose, *A graph theoretic study of the numeric solution of sparse positive definite systems*, in Graph Theory and Computing, Academic Press, New York, 1972.

[10] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 226–283.

[11] D. J. Rose and G. F. Whitten, *A recursive analysis of disection strategies*, in Sparse Matrix Computations, Academic Press, New York, 1976.

[12] R. Schrieber, *A new implementation of sparse gaussian elimination*, ACM TOMS, 8 (1982), pp. 256–276.