SIAM J. MATRIX ANAL. APPL.
Vol. 18, No. 1, pp. 140–158, January 1997

012

AN UNSYMMETRIC-PATTERN MULTIFRONTAL METHOD FOR SPARSE LU FACTORIZATION*

TIMOTHY A. DAVIS† AND IAIN S. DUFF‡

Abstract. Sparse matrix factorization algorithms for general problems are typically characterized by irregular memory access patterns that limit their performance on parallel-vector supercomputers. For symmetric problems, methods such as the multifrontal method avoid indirect addressing in the innermost loops by using dense matrix kernels. However, no efficient LU factorization algorithm based primarily on dense matrix kernels exists for matrices whose pattern is very unsymmetric. We address this deficiency and present a new unsymmetric-pattern multifrontal method based on dense matrix kernels. As in the classical multifrontal method, advantage is taken of repetitive structure in the matrix by factorizing more than one pivot in each frontal matrix, thus enabling the use of Level 2 and Level 3 BLAS. The performance is compared with the classical multifrontal method and other unsymmetric solvers on a CRAY C-98.

Key words. LU factorization, unsymmetric sparse matrices, multifrontal methods

AMS subject classifications. 65F50, 65F05, 65Y15, 65-04

PII. S089547989324690X

NOTATION.

$\mathbf{A}$	original matrix
$\mathbf{A}^k$	undeleted rows and columns of the original matrix at step k
$\mathcal{A}$	Struct($\mathbf{A}$)
$\mathbf{A}'$	active submatrix
$\mathbf{F}$	current frontal matrix
$\mathbf{C}$	contribution block of $\mathbf{F}$
$\mathbf{L}'$, $\mathbf{L}''$	the $\lvert\mathcal{L}'\rvert$ columns of $\mathbf{L}$ computed in $\mathbf{F}$
$\widehat{\mathbf{L}}$	the portion of $\mathbf{L}''$ whose updates have yet to be applied to $\mathbf{C}$
$\mathbf{U}'$, $\mathbf{U}''$	the $\lvert\mathcal{U}'\rvert$ rows of $\mathbf{U}$ computed in $\mathbf{F}$
$\widehat{\mathbf{U}}$	the portion of $\mathbf{U}''$ whose updates have yet to be applied to $\mathbf{C}$
$\mathcal{L}$	sequence of row indices of $\mathbf{F}$ (union of pattern of pivotal columns in $\mathbf{F}$)
$\mathcal{L}'$	pivotal row indices in $\mathcal{L}$
$\mathcal{L}''$	nonpivotal row indices in $\mathcal{L}$ ($\mathcal{L} = \mathcal{L}' \cup \mathcal{L}''$)
$\mathcal{U}$	sequence of column indices of $\mathbf{F}$ (union of pattern of pivotal rows in $\mathbf{F}$)
$\mathcal{U}'$	pivotal column indices in $\mathcal{U}$
$\mathcal{U}''$	nonpivotal column indices in $\mathcal{U}$ ($\mathcal{U} = \mathcal{U}' \cup \mathcal{U}''$)
$\overline{V}$	a list of the first pivotal indices of the factorized frontal matrices
e	an element $e \in \overline{V}$

* Received by the editors November 15, 1995; accepted for publication (in revised form) by J. W. H. Liu February 3, 1996.
http://www.siam.org/journals/simax/18-1/24690.html

† Computer and Information Science and Engineering Department, University of Florida, Gainesville, FL 32611-6120 (davis@cise.ufl.edu). Technical reports, software, and matrices are available via the World Wide Web at http://www.cise.ufl.edu/~davis or by anonymous ftp at ftp.cise.ufl.edu. Support for this project was provided by National Science Foundation grants ASC-9111263 and DMS-9223088 and by CRAY Research, Inc. and Florida State University through the allocation of supercomputer resources. Portions of this work were supported by a postdoctoral grant from CERFACS.

‡ Rutherford Appleton Laboratory, Chilton, Didcot, Oxon 0X11 0QX, England, and European Center for Research and Advanced Training in Scientific Computation (CERFACS), Toulouse, France (isd@letterbox.rl.ac.uk). Technical reports, information on the Harwell Subroutine Library, and matrices are available via the World Wide Web at http://www.cis.rl.ac.uk/struct/ARCD/NUM.html or by anonymous ftp at seamus.cc.rl.ac.uk/pub.

$\mathbf{C}_e$	remaining portions of a previous contribution block ($e < k$)
$\mathcal{L}_e$	row indices of $\mathbf{C}_e$
$\mathcal{U}_e$	column indices of $\mathbf{C}_e$
$\mathcal{C}_j$	element list of column j of $\mathbf{A}'$
$\mathcal{R}_i$	element list of row i of $\mathbf{A}'$
$d_r(i)$	the true degree of row i
$\overline{d_r}(i)$	upper bound of the degree of row i
$d_c(j)$	the true degree of column j
$\overline{d_c}(j)$	upper bound of the degree of column j
$w()$	a work array for computing external column degrees
$\vert \ldots \vert$	number of entries in a matrix or set, or absolute value of a scalar, depending on the context
Struct$(\ldots)$	row indices of entries in a column, or column indices of entries in a row

1. Introduction. Conventional sparse matrix factorization algorithms for general problems rely heavily on indirect addressing. This gives them an irregular memory access pattern that limits their performance on typical parallel-vector supercomputers and on cache-based RISC architectures. In contrast, the multifrontal method of Duff [8], Duff, Erisman, and Reid [9], and Duff and Reid [13, 14] is designed with regular memory access in the innermost loops and has been modified by Amestoy and Duff to use standard kernels [2]. This multifrontal method assumes structural symmetry and bases the factorization on an *assembly tree* generated from the original matrix and an ordering such as minimum degree. The computational kernel, executed at each node of the tree, is one or more steps of LU factorization within a square, dense *frontal matrix* defined by the nonzero pattern of a pivot row and column. These steps of LU factorization compute a *contribution block* (a Schur complement) that is later *assembled* (added) into the frontal matrix of its parent in the assembly tree. Henceforth we will call this approach the *classical* multifrontal method.

Although structural asymmetry can be accommodated in the classical multifrontal method by holding the pattern of $\mathbf{A}+\mathbf{A}^T$ and storing explicit zeros, this can have poor performance on matrices whose patterns are very unsymmetric. If we assume from the outset that the matrix may be structurally asymmetric, the situation becomes more complicated. For example, the frontal matrices are rectangular instead of square, and some contribution blocks must be assembled into more than one subsequent frontal matrix. As a consequence, it is no longer possible to represent the factorization by an assembly tree and the more general structure of an *assembly dag* (directed acyclic graph) [5] similar to that of Gilbert and Liu [20] and Eisenstat and Liu [16, 17, 18] is required. In the current work we do not explicitly use this structure. Since we consider an algorithm that combines the symbolic analysis and numerical factorization, our algorithm for a subsequent numerical factorization (which uses a dag) is beyond the scope of this paper.

We have developed a new unsymmetric-pattern multifrontal approach [4, 5]. As in the symmetric multifrontal case, advantage is taken of repetitive structure in the matrix by factorizing more than one pivot in each frontal matrix. Thus the algorithm can use higher level dense matrix kernels in its innermost loops (Level 3 BLAS [6]). We refer to the unsymmetric-pattern multifrontal method described in this paper as UMFPACK Version 1.0 [4]. A parallel factorize-only version of UMFPACK, based on the assembly dag, is discussed in Hadfield's dissertation [23] and related work [24, 25]. The multifrontal method for symmetric positive definite matrices is reviewed in [27].

Section 2 presents an overview of the basic approach and a brief outline of

the algorithm. We introduce our data structures in the context of a small sparse matrix in section 3, where we describe the factorization of the first frontal matrix. In section 4 we develop the algorithm further by discussing how subsequent frontal matrices are factorized. We have split the discussion of the algorithm into these two sections so that we can define important terms in the earlier section while considering a less complicated situation. Section 5 presents a full outline of the algorithm, using the notation introduced in previous sections. In section 6, we compare the performance of our algorithm with an algorithm based on the classical multifrontal method (MUPS, [2]), and an algorithm based on conventional (compressed sparse vector) data structures (MA48, [15]).

2. The basic approach. Our goal with the UMFPACK algorithm is to achieve high performance in a general unsymmetric sparse factorization code by using the Level 3 BLAS. We accomplish this by developing a multifrontal technique that uses rectangular frontal matrices and chooses several pivots within each frontal matrix. High performance is also achieved through an approximate degree update algorithm that is much faster (asymptotically and in practice) than computing the true degrees. A general sparse code must select pivots based on both numerical and symbolic (fill-reducing) criteria. We therefore combine the analysis phase (pivot selection and symbolic factorization) with the numerical factorization. We construct our rectangular frontal matrices dynamically, since we do not know their structure prior to factorization. An assembly dag is constructed during this analyze–factorize phase. We use the assembly dag in the factorize-only phase, and Hadfield [23] and Hadfield and Davis [24, 25] develop it further and use it in a parallel factorize-only algorithm.

The *active matrix* is the Schur complement of $\mathbf{A}$ that remains to be factorized. At a particular stage, the frontal matrix is initialized through choosing a pivot from anywhere in the active matrix (called a global pivot search) using a Zlatev-style pivot search [29], except that we keep track of upper bounds on the degrees of rows and columns in the active matrix, rather than the true degrees. (The degree of a row or column is simply the number of entries in the row or column.) We call this first pivot the *seed* pivot. Storage for the frontal matrix is allocated to contain the entries in the pivot row and column plus some room for further expansion determined by an input parameter. We denote the current frontal matrix by $\mathbf{F}$ and the submatrix comprising the rows and columns not already pivotal by $\mathbf{C}$, calling $\mathbf{C}$ the contribution block.

Subsequent pivots within this frontal matrix are found within the contribution block $\mathbf{C}$, as shown in Figure 2.1. The frontal matrix grows as more pivots are chosen, as denoted by the arrows in the figure. We assemble contribution blocks from earlier frontal matrices into this frontal matrix as needed. The selection of pivots within this frontal matrix stops when our next choice for pivot would cause the frontal matrix to become larger than the allocated working array. We then complete the factorization of the frontal matrix using Level 3 BLAS, store the LU factors, and place the contribution block $\mathbf{C}$ in a heap. The contribution block is deallocated when it is assembled into a subsequent frontal matrix. We then continue the factorization by choosing another seed pivot and generating and factorizing a new frontal matrix.

It is too expensive to compute the actual degrees of the rows and columns of the active matrix. To do so would require at least as much work as the numerical factorization itself. This would defeat the performance gained from using the dense matrix kernels. Instead, we compute upper bounds for these degrees at a much lower complexity than the true degrees, since they are obtained from the frontal matrix data structures instead of conventional sparse vectors. We avoid forming the union of

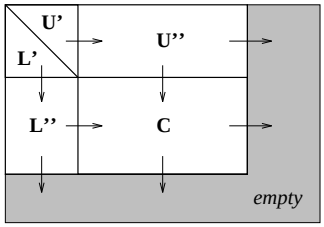

FIG. 2.1. *A rectangular frontal matrix within a larger working array.*

sparse rows or columns which would have been needed were we to compute the filled patterns of rows and columns in the active matrix. We have incorporated a symmetric analogue of our approximate degree update algorithm into the approximate minimum degree (AMD) ordering algorithm [1]. The algorithm produces the same quality of ordering as prior minimum degree ordering algorithms and is typically faster.

The performance we achieve in the UMFPACK algorithm thus depends equally on two crucial factors: this approximate degree update algorithm and the numerical factorization within dense, rectangular frontal matrices. An outline of the UMFPACK algorithm is shown in Algorithm 1. If $\mathbf{A}$ is permuted to block upper triangular form [11], the algorithm is applied to each block on the diagonal. Algorithm 1 consists of initializations followed by three steps, as follows:

ALGORITHM 1 (outline of the unsymmetric-pattern multifrontal algorithm).

```
0:   initializations
     while (factorizing A) do
1:       global pivot search for seed pivot
         form frontal matrix F
         while (pivots found within frontal matrix) do
2:           assemble prior contribution blocks and original rows into F
             compute the degrees of rows and columns in C (the contribution
             block of F)
             numerically update part of C (Level 2 and Level 3 BLAS)
             local pivot search within C
         endwhile
3:       complete the factorization of F using Level 3 BLAS
         place C in heap
     endwhile
```

The initialization phase of the algorithm (step 0) converts the original matrix into two compressed sparse vector forms (row oriented and column oriented [9]) with numerical values $\mathbf{A}$ and symbolic pattern $\mathcal{A}$. Rows and columns are used and deleted from $\mathbf{A}$ and $\mathcal{A}$ during factorization when they are assembled into frontal matrices. At any given step, k say, we use $\mathbf{A}^k$ and $\mathcal{A}^k$ to refer to entries in the original matrix that are not yet deleted. An *entry* is defined by a value in the matrix that is actually stored. Thus all nonzeros are entries but some entries may have the value zero. We use $|\ldots|$ both to denote the absolute value of a scalar and to signify the number of entries in a set, sequence, or matrix. The meaning should always be quite clear from the context.

The true degrees $d_r(i)$ and $d_c(j)$ are the number of entries in row i and column j of the active matrix $\mathbf{A}'$, respectively, but we do not store these. Because the cost

of updating these would be prohibitive, we instead use upper bounds $\overline{d_r}(i)$ (where $d_r(i) \le \overline{d_r}(i)$) and $\overline{d_c}(j)$ (where $d_c(j) \le \overline{d_c}(j)$). However, when a true degree is computed, as in the initialization phase or during the search for a seed pivot, its corresponding upper bound is set equal to the true degree.

3. The first frontal matrix. We will label the frontal matrix generated at stage e by the index e. We now describe the factorization of the first frontal matrix ($e = 1$). This discussion is, however, also applicable for subsequent frontal matrices ($e > 1$) which are discussed in full in section 4 where differences from the case $e = 1$ are detailed.

3.1. Step 1: Perform global pivot search and form frontal matrix. The algorithm performs pivoting both to maintain numerical stability and to reduce fill in. The first pivot in each frontal matrix is chosen using a global Zlatev-style search [29]. A few candidate columns with the lowest upper bound degrees are searched. The number searched is controlled by an input parameter (which we denote by *nsrch* and whose default value is four). Among those *nsrch* columns, we select as pivot the entry a'_{rc} with the smallest approximate Markowitz cost [28], $(\overline{d_r}(r) - 1)(d_c(c) - 1)$, such that a'_{rc} also satisfies a threshold partial pivoting condition [9]

$$|a'_{rc}| \ge u \cdot \max_i |a'_{ic}|, \quad 0 < u \le 1. \tag{3.1}$$

Note that we have the true column degree. The column entries are just the entries in $\mathbf{A}$ since this is the first frontal matrix. When the pivot is chosen its row and column structure define the frontal matrix. If Struct(...) denotes the row indices of entries in a column or column indices of entries in a row, we define $\mathcal{L}$ and $\mathcal{U}$ by $\mathcal{L} = \text{Struct}(\mathbf{A}'_{*c})$ and $\mathcal{U} = \text{Struct}(\mathbf{A}'_{r*})$, the row and column indices, respectively, of the current $|\mathcal{L}|$-by-$|\mathcal{U}|$ frontal matrix $\mathbf{F}$. We partition the sets $\mathcal{L}$ and $\mathcal{U}$ into pivotal row and column indices ($\mathcal{L}'$ and $\mathcal{U}'$) and nonpivotal row and column indices ($\mathcal{L}''$ and $\mathcal{U}''$).

We then assemble the pivot row ($\mathbf{A}^k_{r*}$) and column ($\mathbf{A}^k_{*c}$) from the original matrix into $\mathbf{F}$ and delete them from $\mathbf{A}^k$ (which also deletes them from $\mathcal{A}^k$, since $\mathcal{A}^k$ is defined as Struct($\mathbf{A}^k$)).

We then try to find further pivot rows and columns with identical pattern in the same frontal matrix. This process is called *amalgamation*. *Relaxed amalgamation* does the same with pivots of similar but nonidentical pattern. To permit relaxed amalgamation, $\mathbf{F}$ is placed in the upper left corner of a larger, newly allocated, s-by-t work array. Relaxed amalgamation is controlled by choosing values for s and t through the input parameter g, where $s = \lfloor g|\mathcal{L}| \rfloor$, $t = \lfloor g|\mathcal{U}| \rfloor$, and $g \ge 1$. The default value of this parameter in UMFPACK is $g = 2$.

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 & 0 & a_{14} & a_{15} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & a_{25} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 & 0 & 0 & a_{37} \\ a_{41} & 0 & 0 & a_{44} & a_{45} & a_{46} & 0 \\ 0 & a_{52} & a_{53} & 0 & a_{55} & a_{56} & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{66} & a_{67} \\ a_{71} & a_{72} & 0 & 0 & a_{75} & 0 & a_{77} \end{bmatrix}. \tag{3.2}$$

We use example (3.2) to illustrate our discussion in this section and in section 4. Permutations would needlessly obscure the example, so we assume the pivots in the example matrix are on the diagonal, in order. (Note that this assumption would not

TABLE 3.1
True degrees and degree bounds in example matrix.

i	$d_r(i)$	$\overline{d_r}(i)$	j	$d_c(j)$	$\overline{d_c}(j)$
2	4	5	2	4	4
3	5	5	3	3	3
4	3	3	4	4	4
5	4	4	5	5	6
6	2	2	6	3	3
7	4	5	7	3	3

be true if we performed a global pivot search as in Step 1 since in our example the pivots do not have the lowest possible Markowitz cost.) The first pivot is a'_{11}. We have $\mathcal{L} = \mathcal{L}' \cup \mathcal{L}'' = \{1,2,3,4,7\} = \{1\} \cup \{2,3,4,7\}$ and $\mathcal{U} = \mathcal{U}' \cup \mathcal{U}'' = \{1,4,5\} = \{1\} \cup \{4,5\}$. Let g be 1.25; then the 5-by-3 frontal matrix would be stored in a 6-by-3 array.

3.2. Step 2: Choose further pivots, perform assemblies, and partial factorization. We continue our pivot search within the contribution block $\mathbf{C}$ of the current frontal matrix $\mathbf{F}$ and repeat this for as long as there is sufficient space in the working array.

We use the term *assembly* for the addition of contribution terms or original entries via the extend-add ("$\boxplus$") operator [27]. This operator aligns the row and column index sets of its two matrix or vector operands and then adds together values referenced by the same indices. An *implicit* assembly is one that is mathematically represented by the data structures but computationally postponed. An *explicit* assembly is one that is actually computed. An entry in the active matrix $\mathbf{A}'$ is explicitly assembled if all its contribution terms have been added to it, but this is usually not done and such entries are normally only held implicitly. Pivotal rows and columns are always explicitly assembled.

We now describe the test to determine whether a column can be assembled into $\mathbf{F}$. We scan $\mathcal{A}^k_{*j}$ for each column j in $\mathcal{U}''$. The scan of $\mathcal{A}^k_{*j}$ is stopped as soon as a row $i \notin \mathcal{L}$ is found. If the scan completes without such a row being found, then all row indices in $\mathcal{A}^k_{*j}$ are also in $\mathcal{L}$, and we delete $\mathbf{A}^k_{*j}$ from $\mathbf{A}$ and assemble it into $\mathbf{F}$. If this assembly is done, the true degree of column j is $d_c(j) = \overline{d_c}(j) = |\mathcal{L}''|$. If the scan stops early, we compute the upper bound degree of column j as

$$\overline{d_c}(j) = \min \left\{ \begin{array}{ll} n-k & \text{(the size of } \mathbf{A}'\text{)} \\ |\mathcal{L}''| + (|\mathcal{A}^k_{*j}| - \alpha_j) & \text{(the worst case fill-in)} \end{array} \right\},$$

where k is the current step of Gaussian elimination and α_j is the number of entries scanned in $\mathcal{A}^k_{*j}$ before stopping. For each row i in $\mathcal{L}''$, we scan $\mathcal{A}^k_{i*}$ and compute $\overline{d_r}(i)$ in an analogous manner, where we define β_i as the number of entries scanned in $\mathcal{A}^k_{i*}$ before stopping.

In the example, $\mathbf{A}^k_{*4}$ is assembled into $\mathbf{C}$ and entry a_{44} is deleted from $\mathbf{A}$. The uncomputed true degrees and the degree bounds are shown in Table 3.1. The values of α_j used in constructing the upper bounds were obtained on the assumption that the rows and columns of $\mathcal{A}^k$ are stored in ascending order of row and column indices. We make this assumption only to simplify the example. We have

$$\mathbf{F} = \left\{ \begin{array}{c|cc} & \mathcal{U}' & \mathcal{U}'' \\ \hline \mathcal{L}' & a'_{rc} & \mathbf{A}'_{r*} \\ \mathcal{L}'' & \mathbf{A}'_{*c} & \mathbf{C} \end{array} \right\} = \left\{ \begin{array}{c|ccc} & 1 & 4 & 5 \\ \hline 1 & a'_{11} & a'_{14} & a'_{15} \\ 2 & a'_{21} & 0 & 0 \\ 3 & a'_{31} & 0 & 0 \\ 4 & a'_{41} & a_{44} & 0 \\ 7 & a'_{71} & 0 & 0 \end{array} \right\}.$$

We divide the pivot column $\mathbf{A}'_{*c}$ by the pivot a'_{rc} to obtain the kth column of $\mathbf{L}$, the n-by-n lower triangular factor. The pivot row is the kth row of $\mathbf{U}$, the n-by-n upper triangular factor. Step k of Gaussian elimination is complete, except for the updates from the kth pivot. The counter k is now incremented for the next step of Gaussian elimination. The frontal matrix $\mathbf{F}$ is partitioned into four submatrices, according to the partition of $\mathcal{L}$ and $\mathcal{U}$. We have

$$\mathbf{F} = \left\{ \begin{array}{c|cc} & \mathcal{U}' & \mathcal{U}'' \\ \hline \mathcal{L}' & \mathbf{L}'\mathbf{U}' & \mathbf{U}'' \\ \mathcal{L}'' & \mathbf{L}'' & \mathbf{C} \end{array} \right\} = \left\{ \begin{array}{c|ccc} & 1 & 4 & 5 \\ \hline 1 & u_{11} & u_{14} & u_{15} \\ 2 & l_{21} & 0 & 0 \\ 3 & l_{31} & 0 & 0 \\ 4 & l_{41} & a_{44} & 0 \\ 7 & l_{71} & 0 & 0 \end{array} \right\}.$$

The updates to $\mathbf{C}$ from the $|\mathcal{U}'|$ pivots in $\mathbf{F}$ are not applied one at a time. Instead, they are delayed until there are updates pending from b pivots to allow the efficient use of Level 3 BLAS [6]. On a CRAY C-98, a good value for the parameter b is 16. Let $\widehat{\mathbf{L}}$ and $\widehat{\mathbf{U}}$ denote the portions of $\mathbf{L}''$ and $\mathbf{U}''$, respectively, whose updates have yet to be fully applied to $\mathbf{C}$. If $|\mathcal{U}'| \bmod b = 0$ then the pending updates are applied ($\mathbf{C} = \mathbf{C} - \widehat{\mathbf{L}}\widehat{\mathbf{U}}$). If b were 16, no updates would be applied in our example since $|\mathcal{U}'| = 1$.

We now search for the next pivot within the current frontal matrix. We search the columns in $\mathcal{U}''$ to find a candidate pivot column c that has minimum $\overline{d_c}(c)$ among the columns of $\mathcal{U}''$. We then apply any pending updates to this candidate column ($\mathbf{C}_{*c} = \mathbf{C}_{*c} - \widehat{\mathbf{L}}\widehat{\mathbf{U}}_{*c}$) and compute the candidate column $\mathbf{A}'_{*c}$, its pattern $\mathrm{Struct}(\mathbf{A}'_{*c})$, and its true degree $d_c(c)$. (If the updated candidate column is not selected as a pivot, it is not necessary to update it in Step 3 of the algorithm, discussed below in section 3.3.) We select the candidate pivot row r in $\mathcal{L}''$ with the lowest $\overline{d_r}(r)$ such that a'_{rc} also satisfies the threshold pivoting criterion (equation (3.1)). We compute the pattern $\mathrm{Struct}(\mathbf{A}'_{r*})$ of the candidate pivot row and its true degree $d_r(r)$.

If $d_c(c) > s - |\mathcal{U}'|$ or $d_r(r) > t - |\mathcal{U}'|$ the current work array is too small to accommodate the candidate pivot and we stop the pivot search. Also, if the candidate column has entries outside the current frontal matrix, the threshold pivoting criterion might prevent us from finding an acceptable candidate pivot in $\mathcal{L}''$. In this case also we stop the factorization of the current frontal matrix $\mathbf{F}$. If the candidate pivot a'_{rc} is acceptable, then we let $\mathcal{L} = \mathcal{L} \cup \mathrm{Struct}(\mathbf{A}'_{*c})$ and $\mathcal{U} = \mathcal{U} \cup \mathrm{Struct}(\mathbf{A}'_{r*})$. We repartition $\mathcal{L}$ and $\mathcal{U}$ into pivotal row and column indices ($\mathcal{L}'$ and $\mathcal{U}'$) and nonpivotal row and column indices ($\mathcal{L}''$ and $\mathcal{U}''$) and apply any pending updates to the pivot row ($\mathbf{C}_{r*} = \mathbf{C}_{r*} - \widehat{\mathbf{L}}_{r*}\widehat{\mathbf{U}}$).

In the example, the candidate column (column 4) can fit in the 6-by-3 work array (that is, $d_c(4) = 4 \le s - |\mathcal{U}'| = 6 - 1 = 5$). Suppose a'_{44} does not meet the threshold criterion, and row 7 is selected as the candidate row. The candidate row is, however,

rejected when its true degree is computed (the work array is too small to accommodate row 7, since $d_r(7) = 4 > t - |\mathcal{U}'| = 3 - 1 = 2$).

3.3. Step 3: Complete the factorization of F. After the last pivot has been selected within the current frontal matrix $\mathbf{F}$, we apply any pending updates to the contribution block. ($\mathbf{C} = \mathbf{C} - \widehat{\mathbf{L}}\widehat{\mathbf{U}}$, but we do not need to update the failed candidate pivot column, if any.) The pivot rows and columns in $\mathbf{F}$ are then placed in storage allocated for the LU factors.

The contribution block $\mathbf{C}$ and its pattern $\mathcal{L}''$ and $\mathcal{U}''$ form what we call an *element*. In particular, let $\mathbf{C}_e$ denote the contribution block of element e, and let the pattern of $\mathbf{C}_e$ be $\mathcal{L}_e$ and $\mathcal{U}_e$ (note that $\mathcal{L}_e = \mathcal{L}''$ and $\mathcal{U}_e = \mathcal{U}''$). The contribution block $\mathbf{C}_e$ is placed in a heap for assembly into subsequent frontal matrices.

Initially, all row and column indices in $\mathcal{L}_e$ and $\mathcal{U}_e$ are unmarked. When a row (or column) of $\mathbf{C}_e$ is assembled into a subsequent frontal matrix, the corresponding index is marked in $\mathcal{L}_e$ (or $\mathcal{U}_e$). Element e (which consists of the terms $\mathbf{C}_e$, $\mathcal{L}_e$, and $\mathcal{U}_e$) will refer to unmarked portions only. Element e is deleted when all of its entries are assembled into subsequent frontal matrices. For our example, element e is

$$\left\{ \begin{array}{c} \\ 2 \\ 3 \\ 4 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} 4 & 5 \end{array} \\ \left[\begin{array}{cc} c_{24} & c_{25} \\ c_{34} & c_{35} \\ c_{44} & c_{45} \\ c_{74} & c_{75} \end{array} \right] \end{array} \right\}.$$

We associate with each row (column) in the active matrix an *element list*, which is a list of the elements that hold pending updates to the row (column). We denote the list of elements containing row i as $\mathcal{R}_i$ and the list of elements containing column j as $\mathcal{C}_j$. The element lists contain a local index which identifies which row or column in the element matrix is equivalent to the row or column of the active matrix. This facilitates the numerical assembly of individual rows and columns. For each row i in $\mathcal{L}_e$, we place an element/local-index pair (e, m) in the element list $\mathcal{R}_i$, where row i is the mth entry of $\mathcal{L}_e$. Similarly, for each column j in $\mathcal{U}_e$, we place (e, m) in the element list $\mathcal{C}_j$, where column j is the mth entry of $\mathcal{U}_e$.

Let $\sum^{\Cup}$ denote a summation using the $\Cup$ operator. The active matrix $\mathbf{A}'$ is represented by an implicit assembly of $\mathbf{A}^k$ and the elements in the set $\overline{V}$,

$$\mathbf{A}' = \left(\sum_{e \in \overline{V}}^{\Cup} \mathbf{C}_e \right) \Cup \mathbf{A}^k, \tag{3.3}$$

where $\overline{V} \subseteq \{1 \ldots k-1\}$ is the set of elements that remain after step $k-1$ of Gaussian elimination. All $\Cup$ operations in equation (3.3) are not explicitly performed and are postponed, unless stated otherwise. As defined earlier, the notation $\mathbf{A}^k$ refers to original entries in nonpivotal rows and columns of the original matrix that have not yet been assembled into any frontal matrices.

The element lists allow equation (3.3) to be evaluated one row or column at a time, as needed. Column j of $\mathbf{A}'$ is

$$\mathbf{A}'_{*j} = \left(\sum_{(e,m) \in \mathcal{C}_j}^{\Cup} [\mathbf{C}_e]_{*m} \right) \Cup \mathbf{A}^k_{*j} \tag{3.4}$$

TABLE 3.2
Element lists for example matrix after first frontal matrix.

i	$\mathcal{R}_i$	j	$\mathcal{C}_j$
2	(1,1)	2	-
3	(1,2)	3	-
4	(1,3)	4	(1,1)
5	-	5	(1,2)
6	-	6	-
7	(1,4)	7	-

with pattern

$$\text{Struct}(\mathbf{A}'_{*j}) = \left(\bigcup_{e \in \mathcal{C}_j} \mathcal{L}_e \right) \cup \mathcal{A}^k_{*j}. \tag{3.5}$$

Similarly, row i of $\mathbf{A}'$ is

$$\mathbf{A}'_{i*} = \left(\sum_{(e,m) \in \mathcal{R}_i}^{\Updownarrow} [\mathbf{C}_e]_{m*} \right) \Updownarrow \mathbf{A}^k_{i*} \tag{3.6}$$

with pattern

$$\text{Struct}(\mathbf{A}'_{i*}) = \left(\bigcup_{e \in \mathcal{R}_i} \mathcal{U}_e \right) \cup \mathcal{A}^k_{i*}. \tag{3.7}$$

There is an interesting correspondence between our data structures and George and Liu's quotient graph representation of the factorization of a symmetric positive definite matrix [19]. Suppose we factorize a symmetric positive definite matrix using our algorithm and restrict the pivots to the diagonal. Then $\mathcal{A}^k_{i*} = \mathcal{A}^k_{*i}$, $\mathcal{R}_i = \mathcal{C}_i$, $\mathcal{L}_e = \mathcal{U}_e$, and $\text{Adj}_{\mathcal{G}_k}(x_i) = \mathcal{R}_i \cup \mathcal{A}^k_{i*}$, where x_i is an uneliminated node in the quotient graph $\mathcal{G}_k$. The uneliminated node x_i corresponds to a row i and column i in $\mathbf{A}'$. That is, the sets $\mathcal{R}_i$ and $\mathcal{A}^k_{i*}$ are the eliminated supernodes and uneliminated nodes, respectively, that are adjacent to the uneliminated node x_i. In our terminology, the eliminated supernode $\overline{x}_e$ corresponds to element $e \in \overline{V}$. The set $\mathcal{L}_e$ contains the uneliminated nodes that are adjacent to the eliminated supernode $\overline{x}_e$. That is, $\text{Adj}_{\mathcal{G}_k}(\overline{x}_e) = \mathcal{L}_e$.

After the first frontal matrix on example (3.2), $\overline{V} = \{1\}$ and

$$\mathbf{A}' = \mathbf{C}_1 \Updownarrow \mathbf{A}^k = \left\{ \begin{array}{c} \\ 2 \\ 3 \\ 4 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} 4 & 5 \end{array} \\ \left[\begin{array}{cc} c_{24} & c_{25} \\ c_{34} & c_{35} \\ c_{44} & c_{45} \\ c_{74} & c_{75} \end{array} \right] \end{array} \right\} \Updownarrow \left\{ \begin{array}{c} \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 5 & 6 & 7 \end{array} \\ \left[\begin{array}{ccccc} a_{22} & a_{23} & a_{25} & 0 & 0 \\ a_{32} & a_{33} & 0 & 0 & a_{37} \\ 0 & 0 & a_{45} & a_{46} & 0 \\ a_{52} & a_{53} & a_{55} & a_{56} & 0 \\ 0 & 0 & 0 & a_{66} & a_{67} \\ a_{72} & 0 & a_{75} & 0 & a_{77} \end{array} \right] \end{array} \right\}.$$

Note that column four was deleted from $\mathbf{A}^k$ (refer to section 3.2). It also no longer appears in $\mathcal{A}^k$. The element lists are given in Table 3.2. Applying

equations (3.6) and (3.7) to obtain row two, for example, we obtain

$$\mathbf{A}'_{2*} = [\mathbf{C}_1]_{1*} \Updownarrow \mathbf{A}^k_{2*} = \left\{ \begin{array}{cc} & \begin{array}{cc} 4 & 5 \end{array} \\ 2 & \left[\begin{array}{cc} c_{24} & c_{25} \end{array} \right] \end{array} \right\} \Updownarrow \left\{ \begin{array}{cc} & \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ 2 & \left[\begin{array}{ccc} a_{22} & a_{23} & a_{25} \end{array} \right] \end{array} \right\}$$
$$= \left\{ \begin{array}{cc} & \begin{array}{ccc} 4 & 5 & 2 \quad 3 \end{array} \\ 2 & \left[\begin{array}{ccc} c_{24} & (c_{25} + a_{25}) & a_{22} \quad a_{23} \end{array} \right] \end{array} \right\},$$

$$\mathrm{Struct}(\mathbf{A}'_{2*}) = \mathcal{U}_1 \cup \mathcal{A}^k_{2*} = \{4,5\} \cup \{2,3,5\} = \{4,5,2,3\}.$$

4. Subsequent frontal matrices. We now describe how later steps differ when the element lists are not empty by continuing the example with the second frontal matrix.

4.1. Step 1: Perform global pivot search and form frontal matrix. We compute the *nsrch* candidate pivot columns using equations (3.4) and (3.5). The assembled forms of the unused $nsrch - 1$ candidate columns are discarded. Note that this differs from how we treat an unused candidate column during the local pivot search. Updates to a single unused local candidate column are kept, as discussed in section 3.3. In the example, the next pivot is a'_{22}, with $\mathcal{L} = \mathcal{L}' \cup \mathcal{L}'' = \{2,3,5,7\} = \{2\} \cup \{3,5,7\}$ and $\mathcal{U} = \mathcal{U}' \cup \mathcal{U}'' = \{2,3,4,5\} = \{2\} \cup \{3,4,5\}$. The 4-by-4 frontal matrix is stored in a 5-by-5 array ($g = 1.25$).

4.2. Step 2: Choose further pivots, perform assemblies, and partial factorization. In the example, a second pivot (a'_{33}) is found in the second frontal matrix and so we will repeat this step twice.

As we discussed earlier, computing the true degree, $d_c(j) = |\mathrm{Struct}(\mathbf{A}'_{*j})|$, with equation (3.5) would be very time consuming. A loose upper bound on $d_c(j)$ can be derived if we assume no overlap between $\mathcal{L}$ and each $\mathcal{L}_e$, viz.,

$$d_c(j) \leq \min \left\{ \begin{array}{l} n - k, \\ |\mathcal{L}''| + \overline{d_c}(j), \\ |\mathcal{L}''| + (|\mathcal{A}^k_{*j}| - \alpha_j) + \left(\sum_{e \in \mathcal{C}_j} |\mathcal{L}_e| \right). \end{array} \right.$$

This bound is similar to the bound used in the minimum degree ordering algorithm in MATLAB [21], except that it is used in a symmetric context and thus the diagonal entry is excluded from the summation. To compute this bound for all rows and columns in $\mathbf{C}$ would take time

$$\Theta \left(\sum_{i \in \mathcal{L}''} \beta_i + \sum_{j \in \mathcal{U}''} \alpha_j \right)$$

to scan $\mathcal{A}^k$ and time

$$\Theta \left(\sum_{i \in \mathcal{L}''} |\mathcal{R}_i| + \sum_{j \in \mathcal{U}''} |\mathcal{C}_j| \right)$$

to scan $\mathcal{R}_i$ and $\mathcal{C}_j$. For a single column j, the total time is $\Theta(\alpha_j + |\mathcal{C}_j|)$, or $O(|\mathcal{A}^k_{*j}| + |\mathcal{C}_j|)$, since $\alpha_j \leq |\mathcal{A}^k_{*j}|$. Similarly, the time to compute this loose degree bound for a row i is $\Theta(\beta_i + |\mathcal{R}_i|)$, or $O(|\mathcal{A}^k_{i*}| + |\mathcal{R}_i|)$.

However, a much tighter bound can be obtained in the *same* asymptotic time. The set $\mathcal{L}_e$ can be split into two disjoint subsets: the *external* subset $\mathcal{L}_e \setminus \mathcal{L}$ and the *internal* subset $\mathcal{L}_e \cap \mathcal{L}$, where $\mathcal{L}_e = (\mathcal{L}_e \setminus \mathcal{L}) \cup (\mathcal{L}_e \cap \mathcal{L})$, and "\" is the standard set difference operator. Define $|\mathcal{L}_e \setminus \mathcal{L}|$ as the *external column degree* of element e with respect to $\mathbf{F}$. Similarly, define $|\mathcal{U}_e \setminus \mathcal{U}|$ as the *external row degree* of element e with respect to $\mathbf{F}$. We use the bound

$$d_c(j) \le \overline{d_c}(j) = \min \begin{cases} n-k, \\ |\mathcal{L}''| + \overline{d_c}(j), \\ |\mathcal{L}''| + (|\mathcal{A}^k_{*j}| - \alpha_j) + \left(\sum_{e \in \mathcal{C}_j} |\mathcal{L}_e \setminus \mathcal{L}| \right), \end{cases} \tag{4.1}$$

which is tighter than before since $|\mathcal{L}_e \setminus \mathcal{L}| = |\mathcal{L}_e| - |\mathcal{L}_e \cap \mathcal{L}| \le |\mathcal{L}_e|$. The equation for $\overline{d_r}(i)$ is analogous.

An efficient way of computing the external row and column degrees is given in Algorithm 2. (The algorithm for external row degrees is analogous.) The array w is a work array of size n that is used to compute the external column degrees $|\mathcal{L}_e \setminus \mathcal{L}|$. We actually use a slight variation of Algorithm 2 that does not require the assumption that $w(e) = -1$.

ALGORITHM 2 (computation of external column degrees).

```
assume w(e) = −1, for all e ∈ V̄
for each new row i ∈ L do
    for each element e in the element list R_i of row i do
        if (w(e) < 0) then w(e) = |L_e|
        w(e) = w(e) − 1
    end for
end for
```

The cost of Algorithm 2 can be amortized over all subsequent degree updates on the current front. We use the term "amortized time" to define how much of this total work is ascribed to the computation of a single degree bound, $\overline{d_c}(j)$ or $\overline{d_r}(i)$. Note that in computing these amortized time estimates we actually include the cost of computing the external row degrees within the estimate for the column degree bounds although it is actually the external column degrees that are used in computing this bound. We can amortize the time in this way because we compute the external row and column degrees, and the row and column degree bounds, for all rows and columns in the current frontal matrix.

Relating our approximate degree algorithm to George and Liu's quotient graph, our algorithm takes an amortized time of $O(|\mathcal{A}^k_{*j}| + |\mathcal{C}_j|) = O(|\mathrm{Adj}_{\mathcal{G}_k}(x_j)|)$ to compute $\overline{d_c}(j)$. This correspondence holds only if $\mathcal{A}$ is symmetric and pivots are selected from the diagonal. This is much less than the $\Omega(|\mathrm{Adj}_{G_k}(x_j)|)$ time taken to compute the true degree. The true degree $d_c(j) = |\mathrm{Struct}(\mathbf{A}'_{*j})| = |\mathrm{Adj}_{G_k}(x_j)|$ is the degree of node x_j in the implicitly represented elimination graph, G_k [19]. If indistinguishable uneliminated nodes are present in the quotient graph (as used in [26], for example), both of these time complexity bounds are reduced, but computing the true degree still takes much more time than computing our approximate degree.

We now describe how we compute our degree bound $\overline{d_c}(j)$ in an amortized time of $O(|\mathcal{A}^k_{*j}| + |\mathcal{C}_j|)$. We compute the external column degrees by scanning each e in $\mathcal{R}_i$ for each "new" row i in $\mathcal{L}$, as shown in Algorithm 2. A row or column is new if it did not appear in $\mathcal{L}$ or $\mathcal{U}$ prior to the current pivot. Since $e \in \mathcal{R}_i$ implies $i \in \mathcal{L}_e$, row i must be internal (that is, $i \in \mathcal{L}_e \cap \mathcal{L}$). If Algorithm 2 scans element e, the term $w(e)$

is initialized to $|\mathcal{L}_e|$ and then decremented once for each internal row $i \in \mathcal{L}_e \cap \mathcal{L}$. In this case, at the end of Algorithm 2 three equivalent conditions hold:

1. e appears in the list $\mathcal{R}_i$ for some row i in $\mathcal{L}$,
2. the internal subset $\mathcal{L}_e \cap \mathcal{L}$ is not empty,
3. $w(e) = |\mathcal{L}_e| - |\mathcal{L}_e \cap \mathcal{L}| = |\mathcal{L}_e \setminus \mathcal{L}|$.

If Algorithm 2 did not scan element e in any $\mathcal{R}_i$, then the three following equivalent conditions hold:

1. e does not appear in the list $\mathcal{R}_i$ for any row i in $\mathcal{L}$,
2. the internal subset $\mathcal{L}_e \cap \mathcal{L}$ is empty,
3. $w(e) < 0$.

Combining these two cases, we obtain

$$|\mathcal{L}_e \setminus \mathcal{L}| = \left\{ \begin{array}{ll} w(e) & \text{if } w(e) \geq 0 \\ |\mathcal{L}_e| & \text{otherwise} \end{array} \right\} \text{ for all } e \in \overline{V}. \tag{4.2}$$

To compute the external row degrees of all elements, we scan the element list $\mathcal{C}_j$ for each new column j in $\mathcal{U}$ in an analogous manner (with a separate work array). The total time to compute both the external column degrees (Algorithm 2) and the external row degrees is $\Theta(\sum_{i \in \mathcal{L}''} |\mathcal{R}_i| + \sum_{j \in \mathcal{U}''} |\mathcal{C}_j|)$.

We now describe our combined degree update and numerical assembly phase. This phase uses the external row and column degrees for both the degree update and the numerical assembly. We compute $\overline{d_c}(j)$ and assemble elements by scanning the element list $\mathcal{C}_j$ for each column $j \in \mathcal{U}''$, evaluating $\overline{d_c}(j)$ using equations (4.1) and (4.2). If the external row and column degrees of element e are both zero, then we delete (e, m) from $\mathcal{C}_j$ and assemble $\mathbf{C}_e$ into $\mathbf{F}$. Element e no longer exists. This is identical to the assembly from a child (element e) into a parent (the current frontal matrix $\mathbf{F}$) in the assembly tree of the classical multifrontal method. It is also referred to as *element absorption* [13]. It is too costly at this point to delete all references to the deleted element. If a reference to a deleted element is found later on, it is then discarded. If the external column degree of element e is zero but its external row degree is not zero, then (e, m) is deleted from $\mathcal{C}_j$, column j is assembled from $\mathbf{C}_e$ into $\mathbf{F}$, and column j is deleted from element e. Finally, we scan the original entries $(\mathcal{A}^k_{*j})$ in column j as discussed in section 3.2. If all remaining entries can be assembled into the current frontal matrix, then we perform the assembly and delete column j of $\mathbf{A}^k$. Thus, the amortized time to compute $\overline{d_c}(j)$ is $O(|\mathcal{A}^k_{*j}| + |\mathcal{C}_j|)$. This time complexity does not include the time to perform the numerical assembly.

The scan of rows $i \in \mathcal{L}''$ is analogous. The amortized time to compute $\overline{d_r}(i)$ is $O(|\mathcal{A}^k_{i*}| + |\mathcal{R}_i|)$.

We use the sets $\mathcal{L}_e$ and $\mathcal{U}_e$ for all $e \in \overline{V}$ to represent the nonzero pattern of the active matrix using equations (3.5) and (3.7). Our combined degree update and numerical assembly phase reduces the storage required for this representation. These reductions are summarized below:

1. If $|\mathcal{L}_e \setminus \mathcal{L}| = 0$ and $|\mathcal{U}_e \setminus \mathcal{U}| = 0$ then all of $\mathbf{C}_e$ is assembled into $\mathbf{F}$. Element e and all entries in $\mathcal{L}_e$ and $\mathcal{U}_e$ are deleted. This is the same as the complete element absorption that occurs in the classical multifrontal method. Symbolically, this is also the same as element absorption in a quotient graph-based minimum degree ordering algorithm.
2. If $|\mathcal{L}_e \setminus \mathcal{L}| = 0$ and $|\mathcal{U}_e \setminus \mathcal{U}| \neq 0$ then columns $\mathcal{U}_e \cap \mathcal{U}$ are assembled from $\mathbf{C}_e$ into $\mathbf{F}$. The entries $\mathcal{U}_e \cap \mathcal{U}$ are deleted from $\mathcal{U}_e$.
3. If $|\mathcal{L}_e \setminus \mathcal{L}| \neq 0$ and $|\mathcal{U}_e \setminus \mathcal{U}| = 0$ then rows $\mathcal{L}_e \cap \mathcal{L}$ are assembled from $\mathbf{C}_e$ into

F. The entries $\mathcal{L}_e \cap \mathcal{L}$ are deleted from $\mathcal{L}_e$. An example of this assembly is discussed below.

For pivot a'_{22} in the example, we only have one previous element, element 1. The element lists are shown in Table 3.2. The external column degree of element 1 is one, since $|\mathcal{L}_1| = 4$, and $e = 1$ appears in the element lists of three rows in $\mathcal{L}$. The external row degree of element 1 is zero, since $|\mathcal{U}_1| = 2$, and $e = 1$ appears in the element lists of two columns in $\mathcal{U}$. We have $\mathcal{L}_1 = (\mathcal{L}_1 \setminus \mathcal{L}) \cup (\mathcal{L}_1 \cap \mathcal{L}) = \{4\} \cup \{2, 3, 7\}$ and $\mathcal{U}_1 = (\mathcal{U}_1 \setminus \mathcal{U}) \cup (\mathcal{U}_1 \cap \mathcal{U}) = \emptyset \cup \{4, 5\}$. Rows 2, 3, and 7 (but not 4) are assembled from $\mathbf{C}_1$ into $\mathbf{F}$ and deleted. This reduction and assembly corresponds to case 3, above. Row 2 and columns 2 and 3 of $\mathbf{A}^k$ are also assembled into $\mathbf{F}$. No columns are assembled from $\mathbf{C}_1$ into $\mathbf{F}$ during the column scan, since the external column degree of element 1 is not zero.

We have

$$\mathbf{C}_1 = \left\{ \begin{array}{c|cc} & 4 & 5 \\ \hline & - & - \\ 4 & - & - \\ & c_{44} & c_{45} \\ & - & - \end{array} \right\}, \quad \mathbf{A}^k = \left\{ \begin{array}{c|ccc} & 5 & 6 & 7 \\ \hline 3 & 0 & 0 & a_{37} \\ 4 & a_{45} & a_{46} & 0 \\ 5 & a_{55} & a_{56} & 0 \\ 6 & 0 & a_{66} & a_{67} \\ 7 & a_{75} & 0 & a_{77} \end{array} \right\},$$

and

$$\mathbf{F} = \left\{ \begin{array}{c|c|c} & \mathcal{U}' & \mathcal{U}'' \\ \hline \mathcal{L}' & a'_{rc} & \mathbf{A}'_{r*} \\ \hline \mathcal{L}'' & \mathbf{A}'_{*c} & \mathbf{C} \end{array} \right\} = \left\{ \begin{array}{c|c|ccc} & 2 & 3 & 4 & 5 \\ \hline 2 & a'_{22} & a'_{23} & a'_{24} & a'_{25} \\ \hline 3 & a'_{32} & a_{33} & c_{34} & c_{35} \\ 5 & a'_{52} & a_{53} & 0 & 0 \\ 7 & a'_{72} & 0 & c_{74} & c_{75} \end{array} \right\},$$

where we have marked already assembled parts of element 1 by $-$. The set $\mathcal{L}_1$ is now only $\{4\}$, the other entries (2, 3, and 7) having been deleted. It would be possible to recover this space during the computation but we have chosen not to do so in the interest of avoiding the expense of updating the associated element lists. Note then that these lists refer to positions within the original element.

The assembly and deletion of a row in an element does not affect the external column degree of the element, which is why only new rows are scanned in Algorithm 2. Similarly, the assembly and deletion of a column in an element does not affect the external row degree of the element.

The local pivot search within $\mathbf{F}$ evaluates the candidate column c and row r using equations (3.4), (3.5), and (3.7). In the example, the second pivot a'_{33} is found in the local pivot search. The set $\mathcal{L}$ remains unchanged, but the set $\mathcal{U}$ is augmented with the new column 7. Rows 3 and 7 are assembled from $\mathbf{A}^k$ into $\mathbf{F}$ in the subsequent execution of step 2 for this pivot. No further assembly from $\mathbf{C}_1$ is made.

Step 2 is substantially reduced if there are no new rows or columns in $\mathbf{F}$. No assemblies from $\mathbf{A}^k$ or $\mathbf{C}_e$ can be done since all possible assemblies would have been done for a previous pivot. It is only necessary to decrement $\overline{d_c}(j)$ for all $j \in \mathcal{L}''$ and $\overline{d_r}(i)$ for all $i \in \mathcal{U}''$.

4.3. Step 3: Complete the factorization of F. In the example, the final factorized frontal matrix is

$$\mathbf{F} = \left\{ \begin{array}{c|c} & \begin{array}{cc} \mathcal{U}' & \mathcal{U}'' \end{array} \\ \hline \begin{array}{c} \mathcal{L}' \\ \mathcal{L}'' \end{array} & \left[\begin{array}{c|c} \mathbf{L}'\mathbf{U}' & \mathbf{U}'' \\ \hline \mathbf{L}'' & \mathbf{C} \end{array} \right] \end{array} \right\} = \left\{ \begin{array}{c|c} & \begin{array}{ccc|cc} 2 & 3 & & 4 & 5 & 7 \end{array} \\ \hline \begin{array}{c} 2 \\ 3 \\ \hline 5 \\ 7 \end{array} & \left[\begin{array}{cc|cc|c} u_{22} & u_{23} & u_{24} & u_{25} & 0 \\ l_{32} & u_{33} & u_{34} & u_{35} & u_{37} \\ \hline l_{52} & l_{53} & c_{54} & c_{55} & c_{57} \\ l_{72} & l_{73} & c_{74} & c_{75} & c_{77} \end{array} \right] \end{array} \right\}.$$

Note that $u_{27} = 0$, due to the relaxed amalgamation of two pivot rows with non-identical patterns. Relaxed amalgamation can result in higher performance since more of the Level 3 BLAS can be used. In the small example, the active matrix is represented by the implicit assembly

$$\begin{aligned} \mathbf{A}' &= \mathbf{C}_1 \dotplus \mathbf{C}_2 \dotplus \mathbf{A}^k \\ &= \left\{ \begin{array}{c|cc} & 4 & 5 \\ \hline 4 & \left[\begin{array}{cc} - & - \\ - & - \\ c_{44} & c_{45} \\ - & - \end{array} \right] \end{array} \right\} \dotplus \left\{ \begin{array}{c|ccc} & 4 & 5 & 7 \\ \hline 5 & \left[\begin{array}{ccc} c_{54} & c_{55} & c_{57} \end{array} \right. \\ 7 & \left. \begin{array}{ccc} c_{74} & c_{75} & c_{77} \end{array} \right] \end{array} \right\} \\ &\quad \dotplus \left\{ \begin{array}{c|ccc} & 5 & 6 & 7 \\ \hline 4 & \left[\begin{array}{ccc} a_{45} & a_{46} & 0 \end{array} \right. \\ 5 & \left. \begin{array}{ccc} a_{55} & a_{56} & 0 \end{array} \right. \\ 6 & \left. \begin{array}{ccc} 0 & a_{66} & a_{67} \end{array} \right] \end{array} \right\} \\ &= \left\{ \begin{array}{c|cccc} & 4 & 5 & 6 & 7 \\ \hline 4 & \left[\begin{array}{cccc} a'_{44} & a'_{45} & a'_{46} & 0 \end{array} \right. \\ 5 & \left. \begin{array}{cccc} a'_{54} & a'_{55} & a'_{56} & a'_{57} \end{array} \right. \\ 6 & \left. \begin{array}{cccc} 0 & 0 & a'_{66} & a'_{67} \end{array} \right. \\ 7 & \left. \begin{array}{cccc} a'_{74} & a'_{75} & 0 & a'_{77} \end{array} \right] \end{array} \right\}. \end{aligned}$$

The element lists are shown in Table 4.1.

TABLE 4.1
Element lists for example matrix after second frontal matrix.

i	$\mathcal{R}_i$	j	$\mathcal{C}_j$
4	(1,3)	4	(1,1) (2,1)
5	(2,1)	5	(1,2) (2,2)
6	-	6	-
7	(2,2)	7	(2,3)

5. Algorithm. Algorithm 3 is a full outline of the UMFPACK (Version 1.0) algorithm.

ALGORITHM 3 (unsymmetric-pattern multifrontal algorithm).

0: initializations
$k = 1$
$\overline{V}$ = empty
while $(k \leq n)$ **do**
1: $e = k$

global search for kth pivot: a'_{rc}
$\mathcal{L} = \text{Struct}(\mathbf{A}'_{*c})$
$\mathcal{U} = \text{Struct}(\mathbf{A}'_{r*})$
$s = g|\mathcal{L}|$
$t = g|\mathcal{U}|$
form rectangular frontal matrix $\mathbf{F}$ in an s-by-t work array
do until an exit condition (marked with **) is satisfied
2: *assembly and degree update:*
assemble kth pivot row and column into $\mathbf{F}$
scan element lists and compute external degrees
assemble rows and columns from $\mathbf{A}^k$ into $\mathbf{F}$
assemble contribution blocks into $\mathbf{F}$
compute degree bounds
numerical update:
compute entries of $\mathbf{L}$ ($\mathbf{F}_{*c} = \mathbf{F}_{*c}/a'_{rc}$)
$k = k + 1$
if ($|\mathcal{U}'| \bmod b = 0$) $\mathbf{C} = \mathbf{C} - \widehat{\mathbf{L}}\widehat{\mathbf{U}}$
local pivot search and numerical update of candidates:
** **if** ($|\mathcal{U}''| = 0$) **exit** this loop
find candidate pivot column $c \in \mathcal{U}''$
$\mathbf{C}_{*c} = \mathbf{C}_{*c} - \widehat{\mathbf{L}}\widehat{\mathbf{U}}_{*c}$
if ($\overline{d_c}(c) \neq |\mathcal{L}''|$) assemble column c and compute $d_c(c)$
** **if** ($d_c(c) > s - |\mathcal{U}'|$) **exit** this loop
find candidate pivot row $r \in \mathcal{L}''$
** **if** (r not found) **exit** this loop
if ($\overline{d_r}(r) \neq |\mathcal{U}''|$) assemble row r and compute $d_r(r)$
** **if** ($d_r(r) > t - |\mathcal{U}'|$) **exit** this loop
$\mathcal{L} = \mathcal{L} \cup \text{Struct}(\mathbf{A}'_{*c})$
$\mathcal{U} = \mathcal{U} \cup \text{Struct}(\mathbf{A}'_{r*})$
$\mathbf{C}_{r*} = \mathbf{C}_{r*} - \widehat{\mathbf{L}}_{r*}\widehat{\mathbf{U}}$
enddo
3: *final numerical update and saving of contribution block,* $\mathbf{C}$*:*
save $\mathbf{L}'$, $\mathbf{L}''$, $\mathcal{L}$, $\mathbf{U}'$, $\mathbf{U}''$, and $\mathcal{U}$
$\mathbf{C} = \mathbf{C} - \widehat{\mathbf{L}}\widehat{\mathbf{U}}$
$\mathbf{C}_e = \mathbf{C}$
place $\mathbf{C}_e$ in heap
$\mathcal{L}_e = \mathcal{L}''$
$\mathcal{U}_e = \mathcal{U}''$
delete $\mathbf{F}$
$\overline{V} = \overline{V} \cup \{e\}$
add e to element lists
endwhile

6. Performance results. In this section, we compare the performance of UMFPACK Version 1.0 with MUPS [2] and MA48 [15] on a single processor of a CRAY C-98 (although MUPS is a parallel code). Each method has a set of input parameters that controls its behavior. We used the recommended defaults for most of these, with a few exceptions that we indicate below. All methods can factorize general unsymmetric matrices, and all use dense matrix kernels to some extent [6].

TABLE 6.1
Test matrices.

name	n	$\lvert\mathbf{A}\rvert$	sym.	discipline	comments
GRE 1107	1107	5664	0.000	discrete simul.	computer system
GEMAT11	4929	33185	0.001	electric power	linear programming basis
ORANI678	2529	90158	0.071	economics	Australia
PSMIGR 1	3140	543162	0.479	demography	US county-to-county migration
LNS 3937	3937	25407	0.850	fluid flow	linearized Navier–Stokes
HYDR1	5308	23752	0.004	chemical eng.	dynamic simulation
RDIST1	4134	94408	0.059	chemical eng.	reactive distillation
LHR04	4101	82682	0.015	chemical eng.	light hydrocarbon recovery
LHR71	70304	1528092	0.002	chemical eng.	light hydrocarbon recovery

MA48 [15] supersedes the MA28 code [12]. It first performs an ordering phase that also computes most of the factors but discards them. It then performs the numerical factorization to compute the entire LU factors. When the matrix becomes dense enough near the end of factorization (default of 50% dense), MA48 switches to a dense factorization code.

MUPS performs a minimum degree ordering and symbolic factorization on the nonzero pattern of $\mathbf{A} + \mathbf{A}^T$ and constructs an assembly tree for the numerical factorization phase [2, 8, 9, 14]. During numerical factorization, candidate pivot entries must pass a threshold partial pivoting test similar to equation (3.1), except that the test is by rows instead of by columns. Since the other methods we are comparing perform this test by columns, we factorize $\mathbf{A}^T$ with MUPS and then use the factors of $\mathbf{A}^T$ to solve the original system ($\mathbf{Ax} = \mathbf{b}$). MUPS optionally preorders a matrix so that the diagonal is zero free using a maximum transversal algorithm [7]. MUPS always attempts to preserve symmetry. It does not permute the matrix to block upper triangular form. Note that we do not include symmetric-patterned matrices in our test set, for which MUPS is nearly always faster than UMFPACK.

By default, both UMFPACK and MA48 preorder a matrix to block upper triangular form (always preceded by finding a maximum transversal [7]) and then factorize each block on the diagonal [11]. Off-diagonal blocks do not suffer fill in. This can reduce the work for unsymmetric matrices. We did not perform this preordering, since MUPS does not provide the option. UMFPACK has similar input parameters to MA48, although it does not explicitly include a switch to dense factorization code (each frontal matrix is dense, however). We selected the threshold partial pivoting factor (u) to be 0.1 for all four methods.

The methods were tested on a single processor of a CRAY C-98, with 512 Megawords of memory (8-byte words). Version 6.0.4.1 of the Fortran compiler (CFT77) was used. Each method was given 95Mw of memory to factorize the test matrices, listed in Table 6.1. The table lists the name, order, number of entries ($|\mathbf{A}|$), symmetry, the discipline from which the matrix came, and additional comments. The symmetry is the number of *matched* off-diagonal entries over the total number of off-diagonal entries. An entry, a_{ij} ($j \neq i$), is matched if a_{ji} is also an entry. All matrices are available via anonymous ftp. They include matrices from the Harwell–Boeing Collection [10]. One matrix (LHR71) was so ill conditioned that it required scaling prior to its factorization. The scale factors were computed by the Harwell Subroutine Library routine MC19A [3]. Each row was then subsequently divided by the maximum absolute value in the row (or column, depending on how the method implements threshold partial pivoting). No scaling was performed on the other matrices.

The results are shown in Table 6.2. For each matrix, Table 6.2 lists the numerical

TABLE 6.2
Results.

Matrix	method	factor (sec)	total (sec)	$\|\mathbf{L}+\mathbf{U}\|$ (10^6)	memory (10^6)	op count (10^6)
GRE 1107	UMF.	.07	0.30	.09	.3	9.7
	MA48	.11	0.38	.07	.3	8.1
	MUPS	.13	0.38	.19	.4	26.6
GEMAT11	UMF.	.18	.45	.08	.4	1.0
	MA48	.18	.54	.05	.4	.7
	MUPS	.27	.57	.14	.4	2.8
ORANI678	UMF.	.53	2.07	.12	1.1	7.4
	MA48	.32	1.01	.15	.8	14.2
	MUPS	.61	218.69	.39	13.3	87.6
PSMIGR 1	UMF.	15.62	33.99	6.36	26.4	10194.8
	MA48	14.92	28.86	6.40	20.9	10465.3
	MUPS	14.04	323.15	6.21	26.9	9002.4
LNS 3937	UMF.	.45	1.89	.50	1.4	84.8
	MA48	1.00	3.37	.69	2.2	280.4
	MUPS	.71	1.73	.92	1.2	185.8
HYDR1	UMF.	.24	1.05	.15	.6	4.5
	MA48	.28	.81	.08	.4	.9
	MUPS	.57	1.21	.24	.5	10.7
RDIST1	UMF.	.47	1.53	.49	1.4	37.1
	MA48	1.37	4.78	.41	1.6	27.2
	MUPS	.33	2.01	.28	.7	10.3
LHR04	UMF.	.56	2.51	.39	1.5	30.6
	MA48	1.27	4.25	.34	1.3	25.8
	MUPS	1.03	9.89	1.10	2.3	300.3
LHR71	UMF.	12.26	53.80	10.49	30.2	1294.5
	MA48	51.60	171.66	10.08	36.1	1338.4
	MUPS	-	-	-	> 95.0	-

factorization time, total factorization time, number of nonzeros in $\mathbf{L}+\mathbf{U}$ (in millions), amount of memory used (in millions of words), and floating-point operation count (in millions of operations) for each method. The total time includes preordering, symbolic analysis and factorization, and numerical factorization. The time to compute the scale factors for the LHR71 matrix is not included, since we used the same scaling algorithm for all methods. For each matrix, the lowest time, memory usage, or operation count is underlined. We compared the solution vectors, $\mathbf{x}$, for each method. We found that all four methods compute the solutions with comparable accuracy, in terms of the norm of the residual. We do not give the residual in Table 6.2.

MUPS failed on the LHR71 matrix because of insufficient memory. This is a very ill-conditioned problem that causes MUPS to be unable, on numerical grounds, to choose pivots selected by the analysis. This leads to an increase in fill in and subsequent failure.

We also compared UMFPACK with Gilbert and Peierls' partial pivoting code, GPLU [22], and with SSGETRF, a classical multifrontal method in the CRAY Research Library. The peak performance of GPLU was low primarily because its innermost loops do not readily vectorize (even with the appropriate compiler directives). Since this is a limitation of the code and not a fundamental limitation of the algorithm, we do not report the GPLU results. The results for SSGETRF were roughly comparable with MUPS, except that SSGETRF tended to use slightly less memory than MUPS (sometimes as little as 65% of that of MUPS) and was typically slightly slower than MUPS (sometimes twice as slow as MUPS). We thus do not report

the SSGETRF results since they do not change our overall comparison between the classical multifrontal method (MUPS or SSGETRF) and our unsymmetric-pattern multifrontal method (UMFPACK).

Overall, these results show that the unsymmetric-pattern multifrontal method is a competitive algorithm when compared with the classical multifrontal approach (MUPS) and an algorithm based on more conventional sparse matrix data structures (MA48).

Acknowledgments. We thank Patrick Amestoy, Mario Arioli, Michel Daydé, Theodore Johnson, and Steve Zitney for many helpful discussions; John Gilbert for providing a copy of the GPLU factorization code; and Joseph Liu for providing a copy of the MMD ordering code (used for GPLU). Many researchers provided us with large unsymmetric matrices, a class of matrices that is weak in Release 1 of the Harwell–Boeing Collection. We would also like to thank the referees, whose comments and suggestions improved the presentation of the paper.

REFERENCES

[1] P. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886–905.

[2] P. R. Amestoy and I. S. Duff, *Vectorization of a multiprocessor multifrontal code*, Internat. J. Supercomputer Appl., 3 (1989), pp. 41–59.

[3] A. R. Curtis and J. K. Reid, *On the automatic scaling of matrices for Gaussian elimination*, J. Inst. Math. Appl., 10 (1972), pp. 118–124.

[4] T. A. Davis, *Users' Guide to the Unsymmetric-Pattern Multifrontal Package (UMFPACK, Version* 1.0*)*, Technical Report TR-93-020, CISE Dept., Univ. of Florida, Gainesville, FL, 1993. Version 1.0 has been superseded by Version 2.0. For a copy of Version 2.0, send e-mail to netlib@ornl.gov with the one-line message send index from linalg.

[5] T. A. Davis and I. S. Duff, *Unsymmetric-Pattern Multifrontal Methods for Parallel Sparse LU Factorization*, Technical Report TR-91-023, CISE Dept., Univ. of Florida, Gainesville, FL, 1991.

[6] J. J. Dongarra, J. J. Du Croz, I. S. Duff, and S. Hammarling, *A set of level* 3 *basic linear algebra subprograms*, ACM Trans. Math. Software, 16 (1990), pp. 1–17.

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

[8] I. S. Duff, *Parallel implementation of multifrontal schemes*, Parallel Comput., 3 (1986), pp. 193–204.

[9] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford Univ. Press, London, 1986.

[10] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Users' Guide for the Harwell-Boeing Sparse Matrix Collection (Release* 1*)*, Technical Report RAL-92-086, Rutherford Appleton Laboratory, Didcot, Oxon, England, 1992.

[11] I. S. Duff and J. K. Reid, *An implementation of Tarjan's algorithm for the block triangularization of a matrix*, ACM Trans. Math. Software, 4 (1978), pp. 137–147.

[12] I. S. Duff and J. K. Reid, *Some design features of a sparse matrix code*, ACM Trans. Math. Software, 5 (1979), pp. 18–35.

[13] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.

[14] I. S. Duff and J. K. Reid, *The multifrontal solution of unsymmetric sets of linear equations*, SIAM J. Sci. Comput., 5 (1984), pp. 633–641.

[15] I. S. Duff and J. K. Reid, *MA*48*, a Fortran Code for Direct Solution of Sparse Unsymmetric Linear Systems of Equations*, Technical Report RAL-93-072, Rutherford Appleton Laboratory, Didcot, Oxon, England, 1993.

[16] S. C. Eisenstat and J. W. H. Liu, *Exploiting structural symmetry in unsymmetric sparse symbolic factorization*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 202–211.

[17] S. C. Eisenstat and J. W. H. Liu, *Exploiting structural symmetry in a sparse partial pivoting code*, SIAM J. Sci. Statist. Comput., 14 (1993), pp. 253–257.

[18] S. C. Eisenstat and J. W. H. Liu, *Structural representations of Schur complements in sparse matrices*, in Graph Theory and Sparse Matrix Computation, The IMA Volumes in Mathematics and its Applications, Vol. 56, A. George, J. R. Gilbert, and J. W. H. Liu, eds., Springer-Verlag, Berlin, 1993, pp. 85–100.

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

[20] J. R. Gilbert and J. W. H. Liu, *Elimination structures for unsymmetric sparse LU factors*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 334–354.

[21] J. R. Gilbert, C. Moler, and R. Schreiber, *Sparse matrices in MATLAB: Design and implementation*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 333–356.

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

[23] S. M. Hadfield, *On the LU Factorization of Sequences of Identically Structured Sparse Matrices within a Distributed Memory Environment*, Ph.D. thesis, CISE Dept., Univ. of Florida, Gainesville, FL, 1994; Technical Report TR-94-019, CISE Dept., Univ. of Florida, Gainesville, FL.

[24] S. M. Hadfield and T. A. Davis, *Potential and achievable parallelism in the unsymmetric-pattern multifrontal LU factorization method for sparse matrices*, in Proc. Fifth SIAM Conf. on Applied Linear Algebra, Snowbird, UT, 1994, SIAM, Philadelphia, PA.

[25] S. M. Hadfield and T. A. Davis, *Lost pivot recovery for an unsymmetric-pattern multifrontal method*, Technical Report TR-94-029, CISE Dept., Univ. of Florida, Gainesville, FL.

[26] J. W. H. Liu, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141–153.

[27] J. W. H. Liu, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Review, 34 (1992), pp. 82–109.

[28] H. M. Markowitz, *The elimination form of the inverse and its application to linear programming*, Management Sci., 3 (1957), pp. 255–269.

[29] Z. Zlatev, *On some pivotal strategies in Gaussian elimination by sparse technique*, SIAM J. Numer. Anal., 17 (1980), pp. 18–30.