On the Row Merge Tree for Sparse LU Factorization with Partial Pivoting

L. GRIGORI[1], M. COSNARD[2], and E.G. NG[3] *

[1] *INRIA Rennes, Campus Universitaire de Beaulieu, Avenue du General Leclerc Rennes, 35042, France. email: Laura.Grigori@irisa.fr*

[2] *INRIA Sophia-Antipolis, 2004 Route des Lucioles - BP 93 Sophia Antipolis FR-06902, France. email: Michel.Cosnard@inria.fr*

[3] *Lawrence Berkeley National Laboratory, One Cyclotron Road, Mail Stop 50F-1650 Berkeley, CA 94720-8139, USA. email: EGNg@lbl.gov*

Abstract

We consider the problem of structure prediction for sparse LU factorization with partial pivoting. In this context, it is well known that the column elimination tree plays an important role for matrices satisfying an irreducibility condition, called the strong Hall property.

Our primary goal in this paper is to address the structure prediction problem for matrices satisfying a weaker assumption, which is the Hall property. For this we consider the *row merge matrix*, an upper bound that contains the nonzeros in L and U for all possible row permutations that can later appear in the numerical factorization due to partial pivoting. We discuss the *row merge tree*, a structure that represents information obtained from the row merge matrix. That is, information on the dependencies among the columns in Gaussian elimination with partial pivoting and on structural upper bounds of the factors L and U.

We present new theoretical results that show that the nonzero structure of the row merge matrix can be described in terms of branches and subtrees of the row merge tree. These results lead to an efficient algorithm for the computation of the row merge tree, that uses as input the structure of A alone, and has a time complexity almost linear in the number of nonzeros in A. We also investigate experimentally the usage of the row merge tree for structure prediction purposes on a set of matrices that satisfy only the Hall property. We analyze in particular the size of upper bounds of the structure of L and U, the reordering of the matrix based on a postorder traversal and its impact on the memory hierarchy usage. We note that the practical interest of the row merge tree is limited. For only a small number of matrices in our test set, the row merge tree is a preferred alternative to the column elimination tree.

AMS subject classification: 65F50, 65F05, 68R10

Key words: sparse LU factorization, partial pivoting, structure prediction, column elimination tree, row merge tree.

*The work of the first and the third authors was supported by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy.

1 Introduction

In this paper we consider the problem of structure prediction when solving a sparse system of linear equations $Ax = b$ by Gaussian elimination with partial pivoting, where A is an $n \times n$, nonsingular, nonsymmetric matrix and b is an n-vector.

The *column elimination tree* plays an important role in the structure prediction problem for the sparse LU factorization with partial pivoting, and is used by many solvers implementing this factorization [1, 3, 4, 5, 13]. This tree predicts all the potential column dependencies during Gaussian elimination. It allows upper bounds for the structure of L and U to be computed efficiently. It can also be used to improve memory hierarchy usage if the matrix is permuted according to a postorder traversal. The columns of the permuted matrix can be partitioned so that each group of columns in $L + U$ have similar structures.

It is well known that the dependencies among the columns can be overestimated by the column elimination tree. However it has been shown that when the input matrix satisfies an irreducibility condition, called the strong Hall property [1], the tree provides the tightest information obtainable prior to the numerical factorization of A [15]: for each edge of the column elimination tree, there exists a choice of numerical values of A such that this edge corresponds to a real column dependency.

The upper bound on the structure of L and U predicted using the column elimination tree and the nonzero structure of A represents a structure proposed by George and Ng [11] when the matrix satisfies the strong Hall property. This structure, called the *row merge matrix*, contains the nonzeros in L and U for all possible row permutations that can later appear in the numerical factorization due to partial pivoting. The row merge matrix is also relevant in the structure prediction of QR factorization computed by Householder transformations. Let H be the lower trapezoidal matrix whose columns are the Householder vectors and let R be the upper triangular matrix. When there is no coincidental numerical cancellation, the bound for the structure of L represents the structure of H and the bound for the structure of U represents the structure of R.

Again, when the matrix A satisfies the strong Hall property, the row merge matrix is the tightest possible upper bound for Gaussian elimination with partial pivoting [15]. The connection between the column elimination tree and the row merge matrix is made explicit when considering the following. The column elimination tree is the elimination tree of $A^T A$. Let L_C be the Cholesky factor of $A^T A$. That is, $A^T A = L_C L_C^T$. If A is strong Hall and has a nonzero diagonal, then the upper triangular part of the row merge matrix has the same structure as L_C^T. The tree itself can be computed in time nearly linear in the number of nonzeros of A [14], without explicitly forming the structure of $A^T A$.

Our goal in this paper is to consider matrices satisfying a weaker condition, which is the Hall property [2]. These matrices can be permuted to a block upper triangular form using the Dulmage Mendelsohn decomposition [2, 10, 23]. Each diagonal block satisfies the strong Hall property and its corresponding column elimination tree can be used efficiently, as discussed above. For most matrices, the decomposition contains a large block and several very small blocks on the diagonal [8]. This is one of the main reasons why solvers like UMFPACK [3], SuperLU [4], SuperLU_MT (shared memory version of SuperLU) [5], and the parallel unsymmetric-pattern multifrontal algorithm in [1] do not apply the Dulmage Mendelsohn decomposition. However

[1] A matrix with m rows and n columns has the *strong Hall property* if every set of k columns has nonzeros in at least $k + 1$ rows, for all $1 \leq k < n$ [15].

[2] A matrix with m rows and n columns has the *Hall property* if every set of k columns has nonzeros in at least k rows, for all $0 \leq k \leq n$ [15].

they still use the column elimination tree for structure prediction purposes. With respect to this, it is known that in the Hall case, the upper triangular part of the row merge matrix and the Cholesky factor L_C^T of A^TA can differ, and thus the usage of the column elimination tree can lead to a larger upper bound than the row merge matrix.

In this paper we focus on matrices that satisfy only the Hall property. We assume that A has a zero-free diagonal, and that there are no accidental numerical cancellations during Gaussian elimination.

We determine a tree structure that contains information on the dependencies among the columns in Gaussian elimination and on the structural upper bounds. This information is obtained from the structure of the row merge matrix. We refer to this tree as *the row merge tree*. We present new theoretical results that relate the nonzero structure of rows in the lower triangular part of the row merge matrix to branches of the row merge tree, and the nonzero structure of columns in the upper triangular part of the row merge matrix to subtrees of the row merge tree, respectively. These results lead to an efficient algorithm for computing the row merge tree. It uses as input the structure of A alone, and has a time complexity almost linear in the number of nonzeros in A. We also investigate experimentally the usage of the row merge tree for structure prediction purposes on a set of matrices from a variety of application domains. These matrices satisfy only the Hall property. We observe that for only a limited number of matrices, the row merge tree is a preferred alternative to the column elimination tree. For these matrices, it can be used to determine the counts of the row merge matrix efficiently, thus decreasing the memory requirements. It better predicts similarities between columns and thus a postordering of this tree can increase the usage of BLAS routines.

The paper is organized as follows. Section 2 briefly presents the row merge matrix and background results related to it. In Section 3 we introduce the row merge tree and we use it to characterize the structure of the upper bounds of L and U. This characterization helps us derive a simple and efficient algorithm in Section 4 for its computation. In Section 5 we show that permuting the columns of a matrix according to a postorder on its row merge tree represents an equivalent ordering with respect to the amount of fill in the row merge matrix. Section 6 analyzes experimentally the usage of the row merge tree, and finally, Section 7 concludes the paper.

2 The row merge matrix

Let A be a sparse nonsymmetric $n \times n$ matrix which is to be factorized into LU using Gaussian elimination with partial pivoting, where L is a lower triangular matrix and U is an upper triangular matrix. Denote by a_{ij} the (i,j) element of A. The *bipartite graph* of A, denoted by $H(A)$, has $2n$ vertices (for n rows and n columns of the matrix). There is an edge (i,j) connecting row vertex i to column vertex j whenever a_{ij} is nonzero. A path from vertex i_0 to vertex i_p in H is a sequence of vertices, denoted by $P[i_0 : i_p] = \{i_0, i_1, \ldots, i_p\}$, such that all the vertices are distinct and there is an edge between every two consecutive vertices. The length of this path is p. The intermediate vertices are all the vertices $i_1, \ldots, i_{p-1}$. Whenever is relevant to distinguish between row vertices and column vertices, we use primes on the names of row vertices. For example, i is a column vertex and i' is the row vertex with the same index. The Hall property and the strong Hall property that we have defined for a matrix can be equivalently defined for its bipartite graph.

The row merge matrix, denoted by $A^\times$, is a matrix that contains the nonzeros in L and U for all possible row permutations which can later appear in the numerical factorization due to partial pivoting. It represents an upper bound for the nonzero structure of L and U, and was proposed by George and Ng [11]. To obtain it, at each step of the elimination an upper bound of the structure of L and U is computed. Consider step i and all the rows that are candidate pivot rows at that step. An upper bound of these rows is given by the union of their structures. Thus the structure of each candidate pivot row is replaced by this union. The upper triangular part of the row merge matrix is denoted as $U^{\times}$, and the lower triangular part as $L^{\times}$. The bipartite graph of the row merge matrix is called the *row merge graph*, and is denoted as $H^{\times}(A)$.

An example of construction of the row merge matrix is presented in Figure 1. Its bipartite graph $H(A)$ is displayed at the right of this matrix. At the first step of elimination, rows $1'$ and $5'$ are candidate pivot rows. The union of these two rows is formed, and it replaces the structure of each one of these rows. The fill elements introduced are marked as circled numbers, to visualize the step number that introduced them. This is repeated at each step of the process on the trailing matrix.

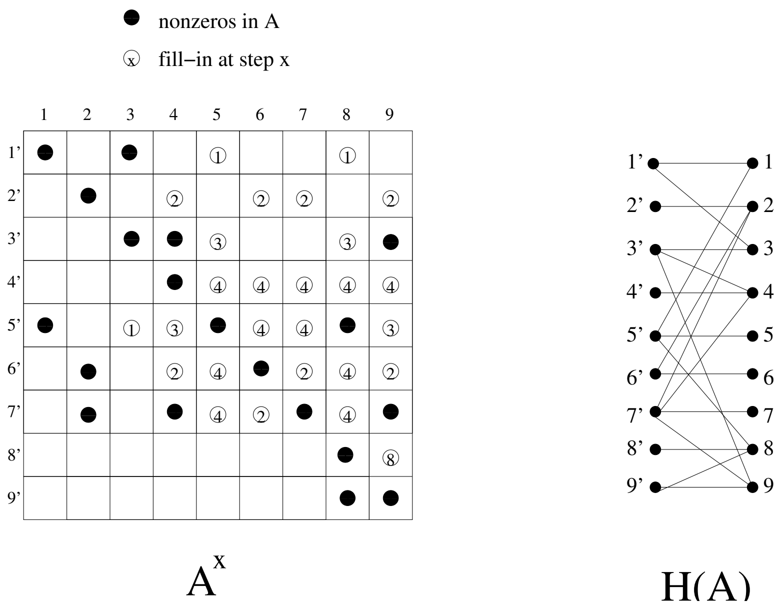


Figure 1: Matrix example A, its row merge matrix $A^{\times}$ and its bipartite graph $H(A)$.

An interesting issue to consider is how tight the bound provided by the row merge graph is for the structure of L and U. When the matrix satisfies the strong Hall property, Gilbert and Ng [15] showed that $H^{\times}(A)$ represents a tight exact bound for the structure of L and U. That is, having a strong Hall graph H, for a given edge (r', c) in its row merge graph $H^{\times}$, there exists a nonsingular matrix A (depending on r' and c) with $H(A) = H$ such that the element in position (r', c) of $L + U$ is nonzero. When the matrix satisfies only the Hall property, it has been shown that the corresponding row merge graph does not represent a tight upper bound on the structure of L and U [16]. Recently, tight bounds were proposed in [17], and finding efficient algorithms for their computation is the subject of future research.

In this paper we refer to *fill elements* as those elements which are zero in the original matrix A and become nonzero in the row merge matrix $A^{\times}$. Similarly, we refer to *fill edges* as those edges which don't belong to the graph $H(A)$ but do belong to the row merge graph $H^{\times}(A)$. These fill edges in the row merge graph $H^{\times}(A)$ are related to paths in the bipartite graph $H(A)$ by the following definition and theorem.

Definition 1 (Gilbert and Ng [15]) A path $P = \{r', c_1, r'_1, c_2, r'_2 \ldots, c_t, r'_t, c\}$ in $H(A)$ is a fill path for LU elimination with partial pivoting if either $t = 0$ or the following conditions are satisfied:

1. $c_k \leq \min(r', c)$ for all $1 \leq k \leq t$.
2. Let c_p be the largest c_k. Then there is some q with $p \leq q \leq t$, $c_p \leq r'_q \leq n$, and the three paths $P[r' : c_p]$, $P[c_p : r'_q]$, $P[r'_q : c]$ are also fill paths in $H(A)$.

The next theorem due to Gilbert and Ng [15] gives a necessary and sufficient condition for fill to occur in the row merge graph $H^\times(A)$.

Theorem 1 (Gilbert and Ng [15]) *Let r' be a row vertex and c be a column vertex of $H(A)$. The edge (r', c) is an edge of the row merge graph $H^\times(A)$ if and only if there is a fill path joining r' and c in $H(A)$.*

For a fill element in position (r', c), Definition 1 identifies the elimination step which introduced it in the row merge matrix, along with the two candidate pivot rows whose union filled that element. Figure 2 illustrates this theorem. In the fill path $P = \{r', c_1, \ldots, c_t, r'_t, c\}$, c_p represents the elimination step at which the element (r', c) is introduced in the row merge matrix. This is due to the fact that r'_q and r' are candidate pivot rows, and the element in position (r'_q, c) is nonzero at this stage of elimination. Thus, during the union of the candidate pivot rows, the element in position (r', c) becomes nonzero in the row merge matrix. As the elements in positions $(r', c_p), (c_p, r'_q)$, and (r'_q, c) can be fill elements from previous elimination steps, the theorem is applied recursively on each of these elements until it arrives at nonzero elements of the original matrix.

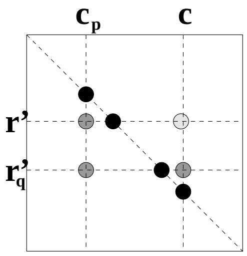


Figure 2: Illustration of Definition 1 for the fill path $P[r' : c] = P[r', \ldots c_p, \ldots r'_q, \ldots c]$.

It follows from Definition 1 that each column involved in the fill path (except the endpoint column c) has at least one off-diagonal nonzero in its $L^\times$ part. This is obvious from the conditions $c_p \leq r'$, $c_p \leq r'_q$, showing that at least the elements in positions (r', c_p) and (r'_q, c_p) are nonzeros.

We illustrate this theorem using our example matrix from Figure 1. Consider the fill element in position $(6', 8)$ and its associated fill path $P[6' : 8] = \{6', 2, 7', 4, 3', 3, 1', 1, 5', 8\}$. In Figure 3 we present a matrix containing only the nonzero elements and fill elements in the row merge matrix leading to the introduction of the nonzero element $u^\times_{68}$. To the right of this matrix we present the fill path in the bipartite graph $H(A)$. In Figure 4 we decompose the fill path $P[6' : 8]$ in sub-fill paths, until we arrive at edges of $H(A)$ corresponding to nonzero elements of the original matrix A. At the root of the tree, as 4 is the column with the biggest index in that path, we can decompose that path in the two paths $P[6' : 4], P[4 : 5']$ and the nonzero element $(5', 8)$. The decomposition in Definition 1 continues recursively on the first two paths. At the bottom of this tree, the leaves are edges of $H(A)$ corresponding to nonzeros of the original matrix A.

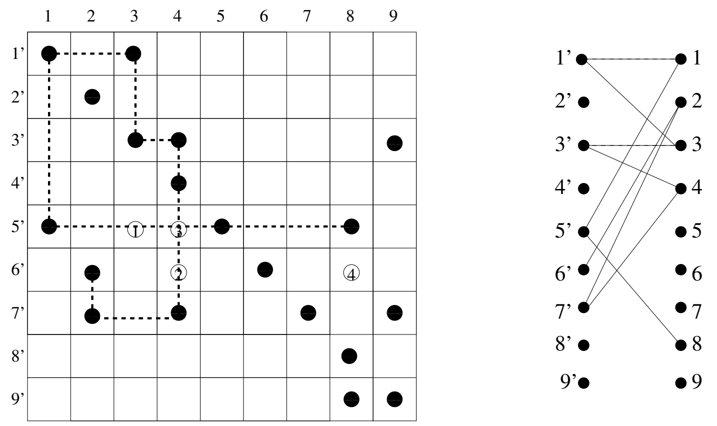


Figure 3: Example of fill element $u^{\times}_{68}$. On the left are nonzero elements and fill elements in $A^{\times}$ which lead to the introduction of $u^{\times}_{68}$. On the right is the fill path $P[6' : 8] = \{6', 2, 7', 4, 3', 3, 1', 1, 5', 8\}$ in $H(A)$, displayed also as a dotted line in the matrix A. The correspondence between the fill elements and subpaths of $P[6' : 8]$ is as follows. The fill element at step 1 corresponds to $P[3 : 5']$. The fill element at step 2 corresponds to $P[6' : 4]$. The fill element at step 3 corresponds to $P[4 : 5']$. The fill element at step 4 corresponds to $P[6' : 8]$.

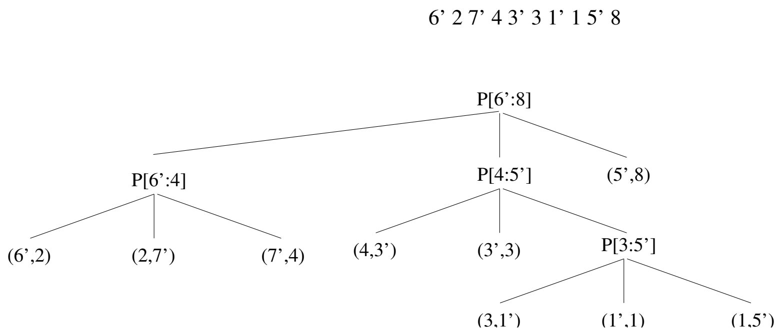


Figure 4: Recursive decomposition of fill path $P[6' : 8] = \{6', 2, 7', 4, 3', 3, 1', 1, 5', 8\}$ using Definition 1.

3 The row merge tree and its relation to the structure of the factors $L^\times$ and $U^\times$

In this section we give the definition of the row merge tree. Then we describe how fill elements in the row merge graph are related to the row merge tree. We also present how the row and column structures of the factors $L^\times$ and $U^\times$ can be characterized using this tree. The characterization will help us derive a simple algorithm in Section 4 to compute the row merge tree.

The row merge tree of A is denoted as $RMT(A)$. In this tree, node k is a direct descendant of node j if node j is the parent of node k; node i is a descendant of node j (node j is an ancestor of node i) if $i < j$ and there is a path from i to j in this tree. The descendants (ancestors) of node j will include j itself.

The structure of row k of A, denoted by $A(k,:)$, is defined as:

$$A(k,:) := \{j \mid a_{kj} \neq 0\}.$$

The number of nonzeros in row k of A is denoted by $|A(k,:)|$. We use f_k to represent the first nonzero in row k of A ($f_k := \min\{j \mid a_{kj} \neq 0\}$). As the matrix is assumed to possess a zero-free diagonal, f_k will be less than or equal to k. The subtree of the row merge tree of A rooted at node k and including all descendants of k in this tree is denoted as $RMT[k]$. The structure of the kth row of $L^\times$ is:

$$L^\times(k,:) := \{j \leq k \mid l^\times_{kj} \neq 0\}.$$

The structure of the kth column of $U^\times$ is:

$$U^\times(:,k) := \{j \leq k \mid u^\times_{jk} \neq 0\}.$$

The row merge tree can be defined from the structure of the row merge matrix $A^\times$ as follows:

Definition 2 The row merge tree of an $n \times n$ matrix A (denoted $RMT(A)$) has n nodes, and j is the parent of k in this tree if and only if $j = \min\{r > k | u^\times_{kr} \neq 0\}$ and $|L^\times(:,k)| > 1$.

By this definition, it is possible to obtain a forest, formed by a set of disjoint trees, but we still call this structure a tree. If j is the root of a tree, then we set $parent[j] = n+1$. Figure 5 presents the column elimination tree and the row merge tree of our example matrix from Figure 1.

When the matrix satisfies the strong Hall property, then the condition $|L^\times(:,k)| > 1$ is always fulfilled. Thus the column elimination tree is equivalent to the row merge tree. However, when the matrix satisfies only the Hall property, the two trees can be different.

The next theorem shows a structural relation which exists between columns of $L^\times$ (respectively rows of $U^\times$) corresponding to nodes lying on a same path of the row merge tree.

Theorem 2 (Shen, Jiao, Yang [24]) *If node j is an ancestor of node k in the row merge tree of A, then $L(j:n,k) \subseteq L(:,j)$ and $U(k,j:n) \subseteq U(j,:)$.*

3.1 Characterization of the structure of $L^\times$

In this section we show that the structure of every row j of $L^\times$ is represented by a branch of the row merge tree $RMT(A)$ that starts at f_j and belongs to $RMT[j]$, the subtree rooted at node j. The same characterization of the structure of $L^\times$, but for matrices satisfying the strong Hall property, was proposed in [12].

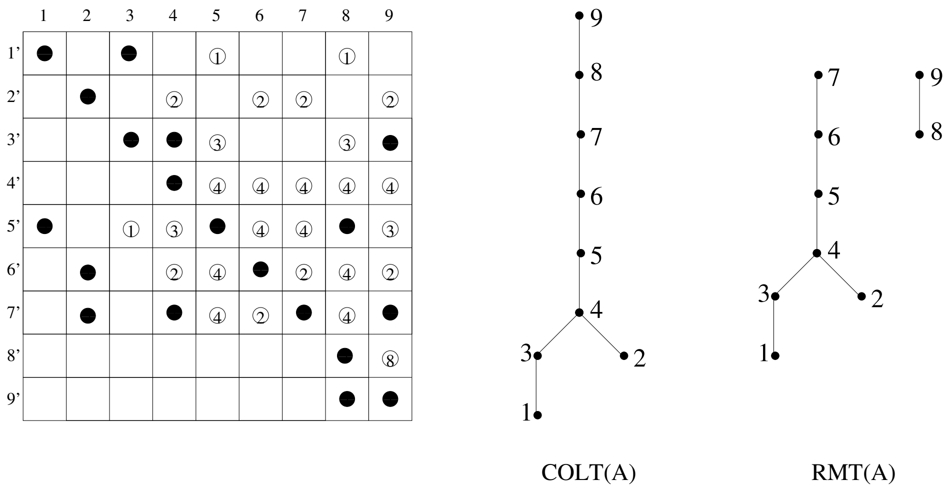


Figure 5: Row merge matrix $A^{\times}$, its column elimination tree $COLT(A)$, and its row merge tree $RMT(A)$.

Lemma 1 *Let $i < k \leq j$ be such that $l^{\times}_{ji} \neq 0$ and $l^{\times}_{jk} \neq 0$ and for every $i < y < k$, $l^{\times}_{jy} = 0$. Then k is the parent of i in the row merge tree of A.*

Proof We prove by contradiction. Suppose that k is not the parent of i in $RMT(A)$. Definition 1 applies with respect to the edges (j, j'), (j', i), and (i, i') of the bipartite graph $H(A)$, and shows that the edge (i', j) is a fill edge of $H(A)$. In other words, the element $u^{\times}_{ij}$ is nonzero. Since the ith column of $L^{\times}$ and the ith row of $U^{\times}$ have each at least one nonzero off-diagonal element, the node i is not the root of a tree in $RMT(A)$. Let q be the parent of i in $RMT(A)$. That means in particular that $u^{\times}_{iq}$ is nonzero. We distinguish two cases: when $q < k$ and when $q > k$.

First, consider the case when $q < k$. Consider step i of construction of the row merge matrix. Note that elements $u^{\times}_{iq}$, $l^{\times}_{ji}$ are nonzero at this step. Thus j is a candidate pivot row, and element $l^{\times}_{jq}$ becomes nonzero. But this contradicts our supposition that all the elements $l^{\times}_{jy}$ are zero, with $i < y < k$.

Second, consider the case when $q > k$. Element $l^{\times}_{jk}$ is nonzero, and Theorem 1 applies, and says that $P[j' : k]$ is a fill path in the bipartite graph $H(A)$. Using Definition 1, we decompose it into three fill paths $P[j' : c_p]$, $P[c_p : r']$, and $P[r' : k]$, where c_p is the largest intermediate column vertex. Consider the fill element $l^{\times}_{jc_p}$ that corresponds to the fill path $P[j' : c_p]$. By our supposition all the elements $l^{\times}_{jy}$ are zero, with $i < y < k$. This shows that $c_p \leq i$. In other words, the fill element $l^{\times}_{jk}$ is introduced either before or at step i of construction of the row merge matrix.

Consider now step i of construction of the row merge matrix. Note that elements $l^{\times}_{jk}$, $u^{\times}_{iq}$ are already nonzero or become nonzero at this step. Row j is a candidate pivot row, and thus element $u^{\times}_{ik}$ becomes nonzero. Since $k < q$, q cannot be the parent of i in the row merge tree of matrix A. And this contradicts our supposition. □

Theorem 3 *For $i \leq j$, $l^{\times}_{ji} \neq 0$ if and only if node i is an ancestor of node f_j in the row merge tree of A, where f_j denotes the column index of the first nonzero entry in row j of A.*

Proof *If Part:* Assume that node i is an ancestor of node f_j, where f_j is the column index of the first nonzero entry in row j of A and $i \leq j$.

If $i = f_j$ then by the definition of f_j, element a_{ji} is nonzero, and thus $l^\times_{ji}$ is also nonzero. Otherwise, Theorem 2 applies and the following holds: $L^\times(i:n, f_j) \subseteq L^\times(:, i)$. As a_{jf_j} is nonzero and $i \leq j$, then $j \in L^\times(:, i)$, hence $l^\times_{ji}$ is nonzero.

Only If Part: Assume that $l^\times_{ji} \neq 0$. Consider the nonzero structure of the jth row of $L^\times$ as $L^\times(j,:) = \{f_j = k_0, k_1, \ldots, k_t = j\}$. Lemma 1 applies to each two consecutive nonzeros, and shows that every node k_i belongs to the path from f_j to j in the row merge tree of A. This shows that i is an ancestor of f_j in the row merge tree of A. □

In particular, Theorem 3 shows that, if some row of $L^\times$ has two different nonzero elements with column indices c_1 and c_2, then the nodes c_1 and c_2 will be in an ancestor-descendant relation in this tree. That is c_1 is descendant of c_2 if $c_1 < c_2$ or ancestor if $c_1 > c_2$.

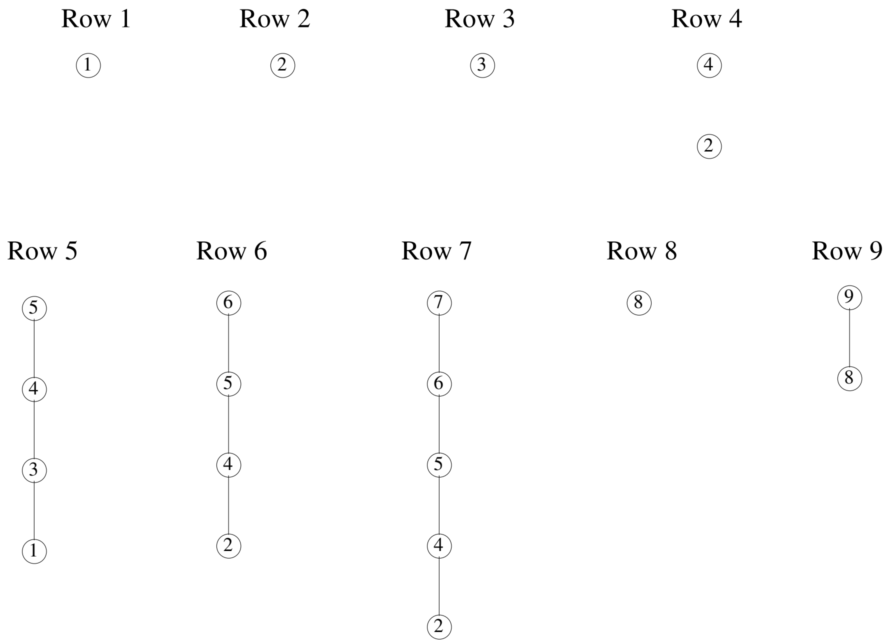


Figure 6: Paths describing the structure of the rows of $L^\times$ for the example matrix in Figure 5.

Theorem 4 shows a structural relation which exists between a node j and its direct descendants $k_1, \ldots, k_r$ in the row merge tree.

Theorem 4 *Let $\{k_1, \ldots, k_r\}$ be the children of node j in $RMT(A)$. Then the structure of column j of $L^\times$ is $L^\times(:, j) := L^\times(:, k_1) \cup \ldots \cup L^\times(:, k_r) \cup A(j:n, j) \backslash \{k_1, \ldots, k_r\}$.*

Proof We prove by contradiction. Suppose there exists an m ($m > j$) such that $l^\times_{mj}$ is nonzero, $m \notin L^\times(:, k)$ for each $k \in \{k_1, \ldots, k_r\}$ and $m \notin A(j:n, j)$.

As $l^\times_{mj}$ is nonzero, j is a node on the path from f_m to m according to Theorem 3. If $f_m = j$, then $a_{mj} \neq 0$. This contradicts our initial supposition. Thus, $f_m < j$. This means there is a $k \in \{k_1, \ldots k_r\}$ which lies on the path from f_m to j in the row merge tree $RMT(A)$. That is, $l^\times_{mk} \neq 0$, resulting in a contradiction. This shows that $L^\times(:, j) := L^\times(j:n, k_1) \cup \ldots \cup L^\times(j:n, k_r) \cup A(j:n, j)$. It is easy to show that for every column k of $L^\times$, where $k \in \{k_1, \ldots, k_r\}$, the row index of the first off-diagonal nonzero is bigger or equal to j. Hence $L^\times(:, j) := L^\times(:, k_1) \cup \ldots \cup L^\times(:, k_r) \cup A(j:n, j) \backslash \{k_1, \ldots, k_r\}$. □

We will use this theorem later to compute the row merge tree. Note that this theorem can be used to count the number of elements in columns of $L^\times$. This is because, as can be easily shown, $L^\times(:, k_1) \cap \ldots \cap L^\times(:, k_r) = \emptyset$.

3.2 Characterization of the structure of $U^\times$

A characterization of the structure of $U^\times$ is not known in the surveyed literature. The next theorems will help define the structure of every column j of $U^\times$.

First we study relations between a fill element in position (r', c), nodes on its associated fill path $P = \{r', c_1, r'_1, c_2, r'_2 \ldots, c_t, r'_t, c\}$, and the row merge tree. When the fill element in position (r', c) belongs to $L^\times$, then node c is a descendant of node r in $RMT(A)$. When it belongs to $U^\times$, then node c can be in a different tree than node r. For example, consider the matrix in Figure 5. The element $u^\times_{78}$ is nonzero in the row merge matrix, but 7 and 8 belong to different trees.

The next lemma shows that node r is an ancestor of all the intermediate column nodes $c_1, \ldots, c_t$ in the fill path $P[r' : c]$. In particular we will see in the proof that all the intermediate nodes $r_1, c_1, \ldots, c_t, r_t$ belong to the same tree as node r.

Lemma 2 *Let (r', c) be a fill edge of the row merge graph $H^\times(A)$ and let $P[r' : c] = \{r', c_1, r'_1, \ldots,$ $c_t, r'_t, c\}$ be its associated fill path. The node r is an ancestor of each intermediate column node $c_1, \ldots c_t$ of the fill path in the row merge tree $RMT(A)$.*

Proof Consider again Definition 1 for the fill path $P[r' : c]$ and Figure 2 which illustrates this theorem. Let c_p be the largest intermediate column node. Item 1 of Definition 1 indicates that $c_p \leq r'$. This means that $l^\times_{rc_p}$ is nonzero. Using the characterization of the structure of a row of $L^\times$, node c_p is a descendant of node r in $RMT(A)$.

Item 2 of Definition 1 indicates that there is some r'_q such that $c_p \leq r'_q$ and the paths $P[r' : c_p], P[c_p : r'_q]$, and $P[r'_q : c]$ are also fill paths. Following the same reasoning as for r, we can conclude that c_p is a descendant of r_q. As $c_p \leq r$ and $c_p \leq r_q$, we can deduce that c_p, r, and r_q are on a same path, such that c_p is a descendant of r and r_q (if $c_p = r_q$ or $c_p = r$, then c_p is a descendant of itself).

We consider now each of the three paths $P[r' : c_p], P[c_p : r'_q]$ and $P[r'_q : c]$ and we show that r is an ancestor of the intermediate column indices in each one of these paths. We assume that the lengths of the paths are larger than 1. Otherwise there is nothing to prove.

First, consider path $P[r' : c_p]$ and let c_{p_1} be the largest intermediate column index in this path ($c_{p_1} \leq r$). By Definition 1, $P[r' : c_{p_1}]$ is a fill path ($l^\times_{rc_{p_1}}$ is nonzero). Thus, node c_{p_1} is a descendant of node r. In particular, as $l^\times_{rc_p}$ is nonzero, we have that node c_{p_1} is a descendant of node c_p in $RMT(A)$.

Second, consider path $P[c_p : r'_q]$. Let c_{p_2} be the largest intermediate column index in this path. Using Definition 1, $P[c_{p_2} : r'_q]$ is a fill path ($c_{p_2} \leq r_q$) and $l^\times_{r_q c_{p_2}}$ is nonzero. As $l^\times_{r_q c_p}$ is also nonzero, we can deduce that c_{p_2} is a descendant of c_p in $RMT(A)$.

Third, consider path $P[r'_q : c]$. Note c_{p_3} the largest intermediate column index on this path. Following the same reasoning as for path $P[c_p : r'_q]$, we conclude that $l^\times_{r_q c_{p_3}}$ is nonzero. We have that $l^\times_{r_q c_p}$ is nonzero. We deduce that c_{p_3} is a descendant of c_p.

In conclusion, c_{p_1}, c_{p_2}, and c_{p_3} are descendants of c_p in $RMT(A)$. As c_p is a descendant of r, we obtain that c_{p_1}, c_{p_2}, and c_{p_3} are descendants of r as well in $RMT(A)$.

Using induction, we conclude that r is an ancestor of all the intermediate column indices $c_1, \ldots c_t$ of the fill path $P[r' : c]$. This completes the proof of the lemma. □

Consider again the fill element at position $(6', 8)$ in the matrix example from Figure 3. Its row merge tree is presented in Figure 5. A fill path is $P[6' : 8] = \{6', 2, 7', 4, 3', 3, 1', 1, 5', 8\}$. We can observe that all the intermediate column indices 2, 4, 3, and 1 are descendants of the node 6 in the row merge tree. Note that node 8 belongs to a different tree.

We use this lemma in the next theorem, which presents a necessary and sufficient condition for fill to occur in $U^\times$ in relation with the row merge tree.

Theorem 5 *For $i < j$, $u^\times_{ij} \neq 0$ if and only if there is a k such that $a_{kj} \neq 0$ and node i is an ancestor of node f_k in the row merge tree $RMT(A)$, where f_k denotes the column index of the first nonzero entry in row k of A.*

PROOF *If Part:* Assume that i is an ancestor of some node f_k, where f_k is the column index of the first nonzero entry in row k of A and a_{kj} is nonzero.

The elements a_{kf_k} and a_{kj} are nonzero. Definition 1 applies with respect to the edges (j, k'), (k', f_k), (f_k, f'_k) and says that (j, f'_k) is a fill edge of the row merge matrix of A. Thus element $u^\times_{f_k j}$ is nonzero.

As i is the ancestor of f_k $(f_k \leq i < j)$, we can apply Theorem 2, and the following holds: $U^\times(f_k, i : n) \subseteq U^\times(i, :)$. As $u^\times_{f_k j}$ is nonzero, then $j \in U^\times(i, :)$. Hence $u^\times_{ij}$ is nonzero.

Only If Part: Assume that $u^\times_{ij} \neq 0$. By Theorem 1, there exists a fill path $P[i' : j] = \{i', c_1, r'_1, \ldots, c_t, r'_t, j\}$ in the graph $H(A)$. Suppose $t \neq 0$, otherwise the proof is trivial. Choose $k = r'_t$.

When decomposing recursively this fill path using Definition 1, the edge (k, j) will be considered at some point of the decomposition. That is, there exist nodes c_p and r'_q such that $c_p \leq k$ and $P[r'_q : c_p]$, $P[c_p : k]$, and $P[k : j]$ are fill paths. The first two paths can be fill paths or can correspond to edges in $H(A)$. The third path is an edge of $H(A)$, corresponding to a nonzero element in the original matrix $(a_{kj} \neq 0)$.

Consider the second path $P[c_p : k]$, with $c_p \leq k$. The element $l^\times_{c_p k}$ is nonzero, and thus c_p is an ancestor of f_k. Lemma 2 on the path $P[i' : j]$ says that c_p is a descendant of node i. Thus we can conclude that node i is an ancestor of node f_k. □

The previous theorem shows that if a_{kj} is nonzero and f_k is the first nonzero in row k, then the structure of the jth column of $U^\times$ includes all nodes on paths from f_k to j or from f_k to the root of the tree including f_k, whichever occurs first.

The next theorem shows that if an element $u^\times_{ij}$ is nonzero, then i belongs to the subtree rooted at j or it can belong to a different tree of root k, where $k < j$. This result will be used in Section 5 to show that permuting the columns of a matrix according to a postorder of its row merge tree is an equivalent ordering with respect to the amount of fill in the row merge matrix.

Theorem 6 *If $u^\times_{ij} \neq 0$, then $i \in RMT[j]$ or $i \in RMT[k]$, where $k < j$ and k is the root of a tree in $RMT(A)$.*

PROOF We will prove by contradiction. Suppose that $i \in RMT[k_1]$, where $parent[k_1] = n+1$ and $k_1 > j > i$. Let m_1, m_2 be ancestors of i in $RMT[k_1]$ such that $parent(m_1) = m_2$, $m_1 < j$ and $m_2 > j$. As $u^\times_{ij} \neq 0$ and $m_1 < j$, Theorem 5 can be applied, so $u^\times_{m_1 j} \neq 0$. This is a contradiction, because by the definition of the row merge tree, $u^\times_{m_1 m_2}$ is the first nonzero element in row m_1 of $U^\times$. □

We conclude that the structure of column j of $U^\times$ can be formed by the following subtrees: a subtree rooted at j; subtrees rooted at $k_1, \ldots, k_r$, where $parent[k_1] = \ldots = parent[k_r] = n + 1$ and $k_1 < j, \ldots, k_r < j$.

In Figure 7 we present the structure of column 9 of our example matrix from Figure 5. This structure is formed by a subtree rooted at 7, having as leaves nodes 2 and 3 (2 is the first nonzero in column 7 and a_{79} is nonzero, a_{39} is nonzero) and a subtree rooted at 9 having as leaf node 8 (8 is the first nonzero in row 9 and a_{99} is nonzero).

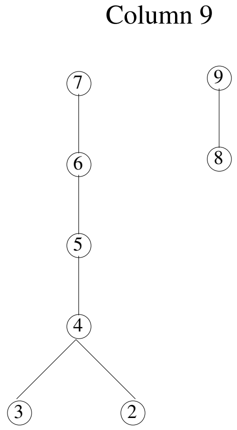


Figure 7: Subforest describing the structure of column 9 of the matrix example in Figure 5.

4 Computing the row merge tree

Algorithm 1 determines the row merge tree from the structure of A alone. That is, it computes the *parent* vector that describes the structure of the row merge tree of A, and it uses a similar approach as the algorithm for computing the symmetric elimination tree [21].

Algorithm 1 uses two additional structures: f_k (defined already in Section 3) denotes the column index of the first nonzero element in row k of A; m_k is the row index of the last nonzero element in column k of $L^\times$ and is initialized as follows:

$$m_k := \max\{j \text{ such that } a_{jk} \neq 0 \text{ and } j \geq k\}$$

At each iteration j, m_j contains the element of maximum index in column j of $L^\times$. This value is necessary because according to Definition 2 if column j of $L^\times$ has only zero off-diagonal elements, then j does not have a parent, i.e. it represents an independent tree in $RMT(A)$. To compute this value, consider the structure of column j of $L^\times$ described in Section 3.1. Let $k_1, k_2, \ldots k_r$ be the immediate descendants of j in $RMT(A)$. The structure of $L^\times(:, j)$ is formed by the union of structures of all columns $L(j : n, k_1), \ldots, L(j : n, k_r)$ and column $A(j : n, j)$. Thus m_j can be updated with the maximum of $m_{k_1}, \ldots, m_{k_r}$ corresponding to the immediate descendants of the node j in the row merge tree.

At iteration j of Algorithm 1, the partial forest corresponding to the first $j - 1$ nodes was formed. The goal of this iteration is to find each node t such that j is the index of the first nonzero in row t of $U^\times$ and $|L^\times(:, t)| > 1$. The t nodes have to be among the roots of the partially formed forest for which $m_t > t$.

Node t can be found as follows. Consider each k such that a_{kj} is nonzero. Let t be the root of the tree containing f_k in the partially formed forest. Theorem 5 applies and indicates that we have $u^{\times}_{tj} \neq 0$. If $m_t > t$, then j is the index of the first nonzero in row t of $U^{\times}$ and thus j becomes the parent of t. If $m_t = t$, then t is the root of the tree in the row merge forest of A and j is not necessarily the index of the first nonzero in row t of $U^{\times}$.

Algorithm 1 Compute the row merge tree.

for column $j := 1$ to n **do**
  $parent[j] := n + 1$
  for $a_{kj} \neq 0$ **do**
    if $f_k = j$ **then** continue **endif**
    find root t of the current tree containing node f_k
    if $t \neq j$ and $m_t > t$ **then**
      $parent[t] := j$
      $m_j := \max(m_j, m_t)$
    end if
  end for
end for

The most expensive part of Algorithm 1 is the search of the root of a subtree. This search can be improved using the union-set problem and techniques as path compression and weighted union presented in [26]. Liu [19, 21] was the first one to exploit these techniques for sparse matrix computations. The basic idea of these techniques is to maintain two forests. The first is the forest of the required row merge tree (the array *parent*) and the second is a compact version of the first which is used to provide the roots of the subtrees quickly. When path compression and balancing are used, the complexity of Algorithm 1 to compute the row merge tree is $O(m\alpha(m, n))$, where m is the number of nonzero elements of the matrix A, n is the number of columns of the matrix A, and $\alpha(m, n)$ is the inverse of Ackermann's function [19, 21, 26] that grows very slowly with m and n.

Consider again the row merge matrix example $A^{\times}$ and its row merge tree $RMT(A)$ in Figure 5. We illustrate the construction of the row merge tree following in Figure 8 the result of each iteration of Algorithm 1.

5 Postordering the row merge tree

To improve cache usage in the numerical factorization step, it is important to group together columns with similar structures in *supernodes*, identifying thus dense submatrices of A. Since the size of supernodes occurring in practice is rather small, several methods are used to increase it.

In the symmetric case, columns with similar structures are made contiguous by permuting the matrix according to a postorder on its elimination tree. This ordering is known to be an equivalent ordering with respect to amount of fill and computation. In the nonsymmetric case, supernodes can be enlarged by permuting the matrix according to a postorder on its column elimination tree [4].

In this section we show that permuting the columns of a matrix according to a postorder on its row merge tree represents an equivalent ordering with respect to the amount of fill in the row merge graph. The permutation is applied to both rows and columns of $A^{\times}$ to preserve the nonzero diagonal. As the row merge tree corresponds to a more accurate upper bound of the structure

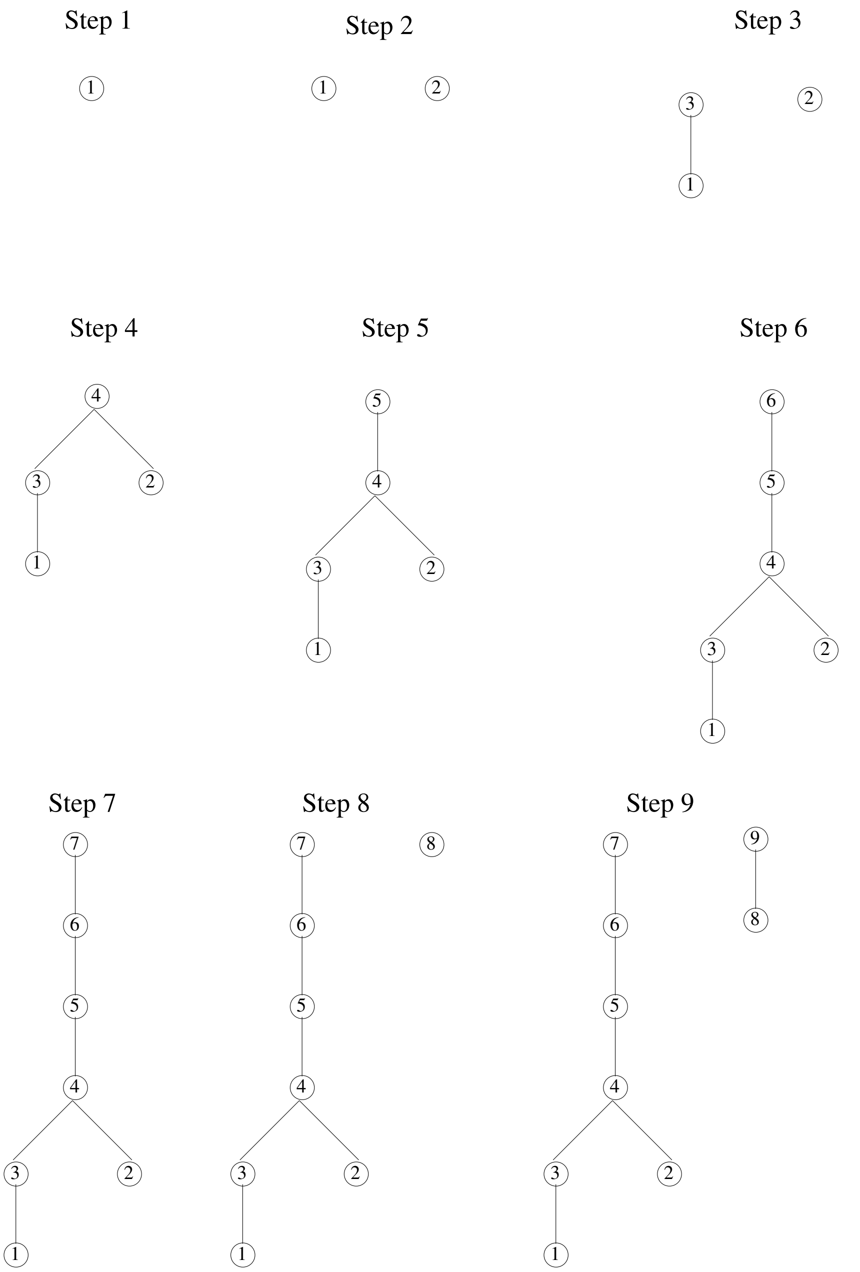


Figure 8: Execution of Algorithm 1 on the example matrix from Figure 1.

of L and U than the column elimination tree, we expect that the supernodes which will actually occur in practice will be of larger size. We will also show that the postordered matrix exhibits a block upper triangular form.

A postordered matrix satisfies the following condition: let $x_1 < \ldots < x_r$ be nodes in the row merge tree of A such that $parent(x_1) = \ldots = parent(x_r) = x$. Then $\forall m, i$ such that $m \in T[x_i], m > x_{i-1}$.

Next, we give an algorithm which takes as input a matrix $A^{\times}$ and its row merge tree and returns a matrix satisfying the above property. This algorithm only helps to prove that the postorder represents an equivalent ordering in terms of fill in the row merge matrix. For the ease of implementation, one would code the postorder as a depth-first traversal of the row merge tree.

The trees in the row merge forest $T_1, \ldots, T_r$ are input as parameters to this algorithm in ascending order of their roots. The number of trees r in the row merge forest is also parameter to this function. The vector *root* is used to determine the root of each subtree of the row merge tree. The postordering is performed through permutations of two consecutive rows and columns x and $x+1$. This allows us to derive a simple proof to show that postordering the row merge tree represents an equivalent ordering with respect to the fill in the row merge graph.

The algorithm iterates over the trees of the forest in descending order of their roots. At each iteration, one tree is postordered as follows. Suppose that Algorithm 2 arrives at the ith iteration, that is, at subtree T_i. First, the algorithm performs permutations such that all nodes in this subtree have labeling bigger than that of the root of subtree T_{i-1}. This ensures that subtree T_i contains all the nodes that should belong to it in the postordered $RMT(A)$. But the subtree itself is not yet postordered. Second, Algorithm 2 is called recursively on the subtrees associated with the direct descendants of the root of T_i.

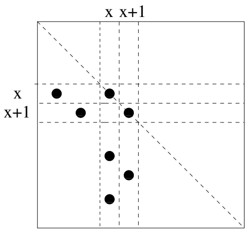
Algorithm 2 Postordering the row merge tree.

```
postorder(T_1, ..., T_r, r)
root[T_0] := 0
for i := r down-to 1 do
  loop
    if ∃x ∈ T_i such that x < root[T_{i-1}] then
      let x be the biggest number with this property
      interchange rows and columns (x, x + 1)
      update parent[] to reflect the interchanges
    else
      break out of the loop
    end if
  end loop
  r' := number of direct descendants of T_i
  if r' ≠ 0 then
    let T'_1, ... T'_{r'} be direct descendants of T_i, root[T'_1] < ... < root[T'_{r'}]
    postorder(T'_1, ..., T'_{r'}, r')
  end if
end for
```

Theorem 7 *The ordering obtained using Algorithm 2 represents an equivalent ordering with respect to the amount of fill in the row merge graph.*

PROOF We will prove this by induction.

Initial phase The structure of the row merge matrix $A^\times$ can be changed only during the interchanging of two consecutive rows/columns in Algorithm 2. Suppose that Algorithm 2 interchanges rows/columns x and $x+1$ and let P_1 describe this permutation. This is illustrated in Figure 9. We show that the amount of fill of the row merge matrix $(P_1^T A P_1)^\times$ is the same as the amount of fill of the permuted row merge matrix $P_1^T A^\times P_1$. For this, we show that during the process of forming the row merge matrix of A, at step x $(x+1)$ the same candidate pivot rows, with the same partial structure are present at step $x+1$ (x) when forming the row merge matrix of $P_1^T A P_1$.



(a)

Figure 9: Illustration for Theorem 7.

Suppose that the row merge tree is formed of the trees $T_1, \ldots, T_r$, and consider that they are in ascending order of their roots. Node x is the largest number in T_r which satisfies the condition $x < root[T_{r-1}]$, and thus node $x+1$ does not belong to T_r. Using the characterization of rows in $L^\times$ from Section 3.1, we have that $L^\times(x, 1:x) \cap L^\times(x+1, 1:x+1) = \emptyset$ and $L^\times(x+1:n, x) \cap L^\times(x+1:n, x+1) = \emptyset$. Thus, interchanging rows (columns) x and $x+1$ does not introduce new candidate pivot rows at steps where x or $x+1$ are themselves candidate pivot rows.

When interchanging rows x and $x+1$, if element $u^\times_{x,x+1}$ is nonzero, then element $l^\times_{x+1,x}$ becomes nonzero, and this element introduces row $x+1$ as a new candidate pivot row at step x of elimination. Thus we have to prove that $u^\times_{x,x+1} = 0$ before interchanging x with $x+1$ for our theorem to be valid. Suppose that $u^\times_{x,x+1}$ is nonzero. Theorem 6 implies that x belongs to either the subtree rooted at $x+1$ or to a different tree of $RMT(A)$ of root $k < x+1$. Since $root[T_r] > x+1$, neither of these conditions is fulfilled, and thus $u^\times_{x,x+1} = 0$. We can conclude that interchanging rows/columns x and $x+1$ does not introduce new candidate pivot rows at steps x, $x+1$ of elimination.

Induction phase We suppose that Algorithm 2 performed a sequence of consecutive rows/columns interchanges, described by P_{m-1}, such that the row merge matrix $(P_{m-1}^T A P_{m-1})^\times$ has the same amount of fill as the row merge matrix $P_{m-1}^T A^\times P_{m-1}$. We consider that Algorithm 2 performs the permutation of two consecutive rows/columns x and $x+1$. Let P_m be the permutation matrix that describes all the permutations performed by Algorithm 2, including the last one. Using a proof identical to the initial phase, we can show that the row merge matrix of $(P_m^T A P_m)^\times$ has the same amount of fill as the permuted row merge matrix $P_m^T A^x P_m$.

Let P be the permutation matrix that describes all the row interchanges performed by Algorithm 2. We have proved that the row merge matrix of $P^T A P$ has the same amount of fill as the permuted row merge matrix $P^T A^x P$. $\square$

To better understand the differences when postordering on the column elimination tree and postordering on the row merge tree, we consider an example matrix in Figure 10. We assume that the pivots used during the numerical factorization are the diagonal elements. The fill elements introduced in L and U are marked as empty circles. We compute its column elimination tree, its row merge tree and postorder according to each of these trees; we provide comments on the outcome.

Figure 10 presents at the right the column elimination tree corresponding to the matrix at the left. In the middle we show the matrix obtained after permuting it according to a postorder on its column elimination tree. Note that even after postordering, this matrix does not have any supernode. One method to increase the supernode size is to apply relaxation, which following some rule, allows a certain number of zero elements in a supernode, and thus becomes a *relaxed supernode*; in this way, the columns of L will not have exactly the same structure. Note that in this simple example, if we consider three particular zero elements as nonzeros (denoted as artificial nonzeros in Figure 10), it is possible to partition the matrix into four relaxed supernodes.

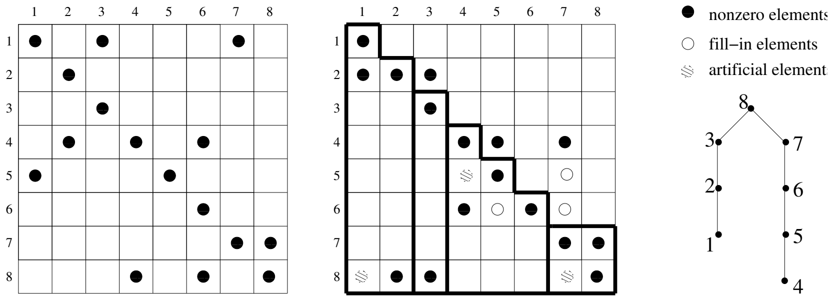


Figure 10: Example matrix B (on the left) and its column elimination tree (on the right). Matrix obtained from B (in the middle) after postordering on its column elimination tree, supernode relaxation and partitioning.

Figure 11 illustrates the same matrix with its row merge tree. In the middle we present the matrix obtained after a postorder traversal of this tree. Note that the matrix has a block upper triangular form when using the row merge tree. Consider that the 4 particular zero elements (denoted as artificial nonzeros in Figure 11) can be considered nonzero. It is possible to obtain 3 supernodes that are larger than the supernodes of matrix in Figure 10.

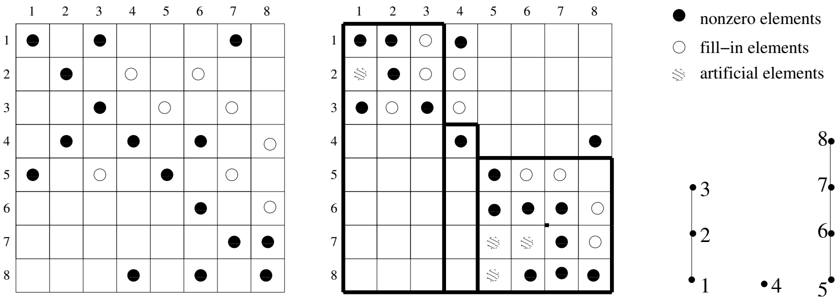


Figure 11: Example matrix B (on the left) and its row merge matrix (on the right). Matrix obtained from B (in the middle) after postordering on its row merge tree, supernode relaxation and partitioning.

In general, let us consider the lower triangular part $L^{\times}$ of the row merge matrix of the postordered matrix P^TAP. For each nonzero element $l_{ij}^{\times}$, Theorem 3 implies that i and j belong to a same tree of $RMT(P^TAP)$. Since the nodes are ordered according to a postorder of the row merge tree, every nonzero element of $L^{\times}$ belongs to a diagonal block, where the nodes of the diagonal block correspond to a tree of $RMT(P^TAP)$. This shows that the postordered matrix exhibits a block upper triangular form.

6 Experimental results

In this section we analyze experimentally the usage of the row merge tree. For this, we consider different approaches that can be employed for the structure prediction problem of the sparse LU factorization with partial pivoting, and discuss their advantages and drawbacks on a set of real world matrices.

The goal of our experiments is two-fold. First, we want to compare the size of the bounds $L_{rmt}^{\times}$ and $U_{rmt}^{\times}$ obtained based on the row merge tree and the size of the bounds $L_{colt}^{\times}$ and $U_{colt}^{\times}$ obtained based on the column elimination tree. These bounds represent upper bounds of the nonzero structure of L and U. This comparison is also relevant for the QR factorization, because the bounds $L_{rmt}^{\times}$ and $U_{rmt}^{\times}$ are equal to the structure of the factors from the orthogonal factorization, when there is no coincidental numerical cancellation. Second, we want to study the impact of using the row merge tree instead of the column elimination tree during the numerical LU factorization. Two aspects are discussed. One aspect considers the size of the factors L and U obtained in the two approaches. In particular, this allows us to evaluate the gap between the estimated upper bounds and the real memory needs of the sparse LU factorization with partial pivoting. Another aspect considers the size of the supernodes and the usage of BLAS routines during the numerical LU factorization.

We use matrices of medium and large sizes from a variety of application domains. These matrices were obtained from the Harwell-Boeing Collection and from the ftp site maintained by Tim Davis of the University of Florida. For all the tests we use Liu's multiple minimum degree algorithm [18] on the structure of A^TA to preorder the columns of A. The runtimes were obtained on a single processor of Cray T3E-900 machine at NERSC. Each node in this machine has a clock rate of 450 MHz and 256 Mbytes memory.

Consider an input matrix A. One approach to factorize it consists of applying the Dulmage Mendelsohn decomposition to transform the matrix in a block upper triangular form such that each diagonal block satisfies the strong Hall property. Thus the column elimination tree can be used for structure prediction purposes. To evaluate this approach, we have used the algorithm presented in [23] to compute the Dulmage Mendelsohn decomposition.

Another approach is to ignore the Dulmage Mendelsohn decomposition on the input matrix. One computes the factorization of the matrix A, and still uses the column elimination tree as the main tool for structure prediction. SuperLU [4] solver and its shared memory version SuperLU_MT [5] are based on this approach. Before the symbolic and numeric computation of the factors, the column elimination tree of matrix A is computed, and the columns of matrix A are permuted according to a postorder on this tree. In the shared memory version, the column elimination tree is also used to guide the parallel execution, and its associated column counts are used to allocate memory. UMFPACK4 [3] uses a slightly different approach. First, the matrix is preprocessed to find a permutation of rows and columns that identifies singletons. That is, the permutation orders first columns that have only zero elements below the diagonal, then rows that have only zero elements in the upper triangular part. This corresponds to columns of L that have

Matrix	Order	$\lvert A \rvert$	Sym.	Maximum matching MC21	Compute coletree	Compute rmtree	Maximum matching BTF	Fine decomp. BTF
bayer02	13935	63679	0.00	0.209	0.035	0.044	0.054	0.055
bayer04	20545	159082	0.00	0.529	0.058	0.074	0.84	0.062
gemat11	4929	33185	0.00	0.027	0.013	0.017	0.024	0.015
graham1	9035	335504	0.72	0.688	0.050	0.059	2.8	0.058
hydr1	5308	23752	0.00	0.072	0.009	0.012	0.058	0.013
jpwh991	991	6027	0.94	0.001	0.002	0.002	0.0007	0.0022
lhr71	70304	1528092	0.00	2.540	0.426	0.530	49.	0.31
lns_3937	3937	25407	0.85	0.194	0.009	0.011	0.11	0.012
lnsp3937	3937	25407	0.85	0.085	0.008	0.010	0.069	0.012
mahindas	1258	7682	0.02	0.007	0.002	0.002	0.002	0.0025
onetone1	36057	341088	0.07	4.571	0.163	0.211	11.	0.16
onetone2	36057	227628	0.11	4.624	0.134	0.170	5.2	0.12
orani678	2529	90158	0.07	0.033	0.013	0.015	0.18	0.012
sherman3	5005	20033	1.00	0.001	0.008	0.010	0.0019	0.011
sherman5	3312	20793	0.74	0.001	0.005	0.006	0.0017	0.0071
shyy161	76480	329762	0.73	0.078	0.203	0.251	0.029	0.22
tols4000	4000	8784	0.59	0.006	0.003	0.004	0.0017	0.0051
twotone	120750	1224224	0.24	93.921	0.543	0.986	117.	0.51
west2021	2021	7353	0.00	0.004	0.002	0.003	0.015	0.0036

Table 1: Benchmark matrices and their characteristics. Runtimes (in seconds) of the maximum matching routine MC21, the column elimination tree algorithm, the row merge tree algorithm and the Dulmage Mendelsohn decomposition (denoted as BTF and divided into time for maximum matching and time for fine decomposition).

only the diagonal element nonzero and rows of U that have only the diagonal element nonzero. And these singletons can be eliminated without causing any fill in the remaining submatrix S. The column elimination tree is used then for structure prediction purposes only on the remaining submatrix S. The parallel unsymmetric-pattern multifrontal solver described in [1] also uses the column elimination tree for structure prediction purposes. Before the computation of the factors, the tree and its associated count bounds for the columns of L and rows of U are used to detect supernodes. During the computation of the factors, the tree is used to exploit parallelism.

We study the usage of the row merge tree as an alternative to the column elimination tree for matrices that are not decomposed using the Dulmage Mendelsohn decomposition. When either the row merge tree or the column elimination tree is used, in our experiments the rows are first permuted such that the matrix has nonzeros on its main diagonal. We use the subroutine MC21A for this step, available in the Harwell library and presented in [6, 7]. We use the algorithm presented in [14] to compute the column elimination tree.

We consider matrices for which Dulmage Mendelsohn decomposition yields a nontrivial block upper triangular form. These matrices and their characteristics are presented in the first four columns of Table 1. The fifth column presents the time spent to find the row permutations for a nonzero diagonal (during the maximum matching algorithm MC21A). The sixth and the seventh columns present the times spent to compute the column elimination tree and to compute the row merge tree, respectively. We can remark that the run time of the row merge tree is slightly higher than that of the column elimination tree. This is due mainly to the computation of the maximum element in each column of $L^{\times}$. This computation has to take into account the row permutations from the maximum matching algorithm, while the column elimination tree algorithm is independent of any row permutations.

The algorithm to compute the block triangular form of a matrix [23], referred to as BTF, considers rectangular matrices and has three different phases. The first phase finds a maximum matching in the bipartite graph $H(A)$. The second phase, called coarse decomposition, decomposes the matrix in three submatrices A_h, A_s and A_v such that A_h is underdetermined, A_s is square and A_v is overdetermined. The third phase, called fine decomposition, further decomposes submatrices A_h, A_s and A_v in block diagonal forms for A_h and A_v and block upper triangular form for A_s. The last two columns of Table 1 present the results obtained when applying this algorithm on our test matrices. As we consider only square matrices in this paper, we present only the time to compute a maximum matching and a fine decomposition. We can remark that, in general, the maximum matching step takes a longer time than the fine decomposition step. For some of the matrices, the difference can be very large.

Comparing results in Table 1, we can observe that the time to compute the row merge tree is always smaller than the sum of the time to do a fine decomposition plus the time to compute the column elimination tree.

6.1 Impact on structure estimation

We restrict our attention to a comparison between the row merge tree and the column elimination tree when they are used to predict upper bounds of the structure of L and U. To compute the number of nonzeros in the bounds $L^{\times}_{colt}$ and $U^{\times}_{colt}$ based on the column elimination tree, we use the algorithms presented in [14]. To compute the number of nonzeros in $L^{\times}_{rmt}$ and $U^{\times}_{rmt}$ based on the row merge tree, we employ the approach described in [16].

For many matrices in our test set, the usage of the row merge tree leads to upper bounds 1% smaller than the upper bounds predicted by the column elimination tree. We ignore these

Matrix		Based on column etree $\lvert L_{colt}\rvert$	$\lvert U_{colt}\rvert$	Based on row merge tree $\lvert L_{rmt}\rvert$	$\lvert U_{rmt}\rvert$	Percentage modification for $\lvert L\rvert$	for $\lvert U\rvert$
bayer04	BOUND	284267	669726	267376	616226	5.9	8.0
	REAL	209338	376356	210297	377734	-0.5	-0.4
hydr1	BOUND	31458	79986	30627	78972	2.6	1.3
	REAL	29357	46363	29433	46255	-0.3	0.2
lhr71	BOUND	3379191	6881650	3356635	6728470	0.7	2.2
	REAL	3078240	4365396	3061621	4359903	0.5	0.1
mahindas	BOUND	20112	50382	16027	40926	20.3	18.8
	REAL	8128	14389	8111	14844	0.2	-3.2
orani678	BOUND	650072	1079969	647999	1037610	0.3	3.9
	REAL	614372	160249	614372	172729	0.0	-7.8
tols4000	BOUND	6061	11133	4276	8753	29.4	21.4
	REAL	4224	8648	4224	8648	0.0	0.0
twotone	BOUND	8529459	23629754	7247851	17901697	15.0	24.2
	REAL	6225802	13276105	6228882	13280833	-0.1	-0.1
west2021	BOUND	8026	19422	8009	18310	0.2	5.7
	REAL	6277	8980	6256	9001	0.3	-0.2

Table 2: Impact of the row merge tree on structure estimation. In the lines $BOUND$, $|L_{colt}|$ and $|U_{colt}|$ ($|L_{rmt}|$ and $|U_{rmt}|$) denote the size of the upper bounds computed based on the column elimination tree (row merge tree). In the lines $REAL$, $|L_{colt}|$ and $|U_{colt}|$ ($|L_{rmt}|$ and $|U_{rmt}|$) denote the factors computed based on the column elimination tree (row merge tree). Their percentage modification is computed as $\frac{|L_{colt}|-|L_{rmt}|}{|L_{colt}|} * 100$ and $\frac{|U_{colt}|-|U^{\times}_{rmt}|}{|U^{\times}_{colt}|} * 100$.

Matrix	Based on column etree #snodes	#flops (10^6)	%flops DGEMV	Based on row merge tree #snodes	#flops (10^6)	%flops DGEMV
bayer04	17825	18.90	69.22	17562	18.86	69.20
hydr1	4826	0.62	4.46	4819	0.62	5.20
lhr71	48403	485.31	65.02	48357	478.78	65.20
mahindas	1171	0.58	56.23	1161	0.58	55.16
orani678	1431	30.88	76.43	1427	30.93	76.39
sherman3	3576	42.91	79.86	3576	42.91	79.86
tols4000	3981	0.03	9.71	3981	0.03	9.71
twotone	77096	12034.32	78.18	75204	12032.45	78.18
west2021	1857	0.04	8.15	1833	0.04	9.70

Table 3: Impact of the row merge tree on numerical factorization. $\#snodes$ denotes the number of supernodes, $\#flops$ denotes the number of floating point operations performed during numerical factorization, and $\%flops\ DGEMV$ denotes the percentage of floating point operations performed in the matrix-vector multiply routine DGEMV.

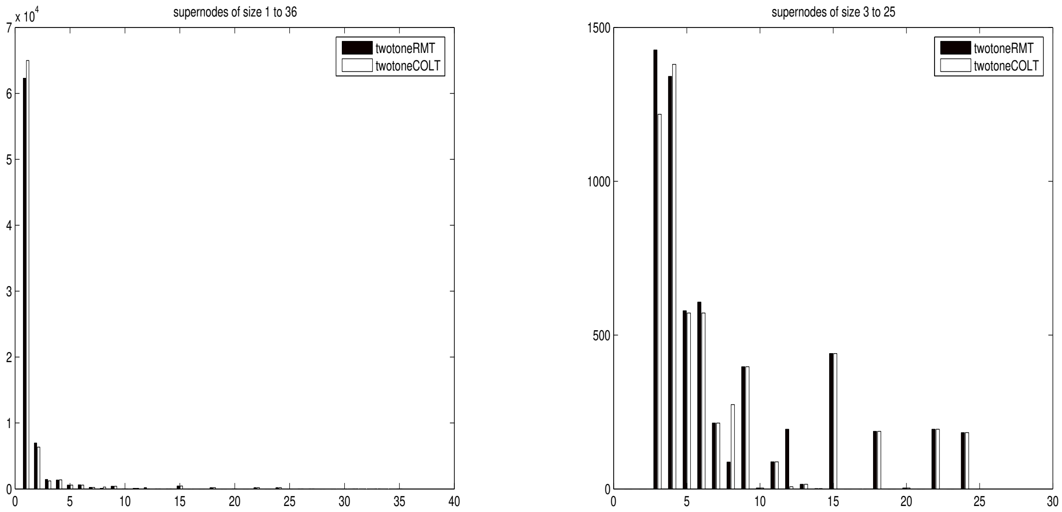


Table 4: Size of supernodes for matrix twotone.

matrices in the sequel of the paper. Table 2 presents only those matrices for which the difference is bigger than 1%. The lines denoted $BOUND$ report the upper bounds of the structure of L and U computed using the column elimination tree (third and fourth columns) and the row merge tree (fifth and sixth columns). We observe that for matrix tols4000, the reduction is as large as 29.4%. For twotone matrix, which is a large, very sparse, and very unstructured matrix, it leads to 24% smaller upper bounds. This shows that the row merge tree can effectively reduce the sizes of the upper bounds of the structure of L and U in some cases.

We also compare the size of the bounds with the real size of the factors L and U. The lines denoted $REAL$ report the size of the factors L and U resulting from the LU factorization with row interchanges. The size of the factors differs slightly when using the column elimination tree (third and fourth columns) and when using the row merge tree (fifth and sixth columns). We observe that the size of the factor L is better estimated than the size of the factor U. For matrix orani678, the bound for U severely overestimate the size of the factor U. But for most of the matrices, the bound is less than two times larger than the real size of the factors. For some matrices, as lhr71, tols4000, the bound is close to the real size of the factors.

6.2 Impact on numerical factorization

To evaluate the impact of the row merge tree on the numerical factorization, we use SuperLU [4] as follows. We have replaced the column elimination tree with the row merge tree in this solver. SuperLU uses a supernodal left-looking approach. At each step of the factorization, a column of the factors L and U is computed using the previously computed supernodes. The most time consuming part in this computation is the update of the current column j by previous supernodes k in the operation $A(i,j) := A(i,j) - L(i,k) \times U(k,j)$. Since L is partitioned in supernodes, each supernode k is represented as a dense matrix. This operation is usually performed through a call to the dense matrix-vector multiplication routine DGEMV in Level 2 BLAS. In SuperLU, when the size of the supernode k or the size of the vector $U(k,j)$ is too small, scalar operations are employed. More generally, when the size of the supernodes involved in this operation is too small, the performance of DGEMV can drop significantly.

We restrict our attention only to the subset of matrices used in Table 2. We report in Table 3 the number of supernodes ($\#snodes$) , the number of floating point operations performed during the numerical factorization ($\#flops$) and the percentage of the floating point operations performed in the matrix-vector multiply DGEMV. To ensure that DGEMV operation uses data that fits in cache, SuperLU controls the maximum size of a supernode by a parameter, which we set at 100.

We can observe that for most of the matrices, the number of supernodes decreases when using the row merge tree. This leads to a larger average size of the supernodes. We expect the size of supernodes to increase when relaxation is used. We did not experiment this option. When the column elimination tree and the row merge tree have almost the same structure, the number of supernodes is comparable in both approaches.

For most of the matrices, the decrease in the number of supernodes leads to some increase in the usage of BLAS2 routines. The gain is more important for matrices like hydr1 (17%), west2021 (19%), for which the percentage of usage of BLAS2 routines is very small. Hence, even a small increase in the average size of supernodes has a bigger impact on the usage of BLAS2 routines.

The only exception is matrix twotone, for which we observe an important decrease in the number of supernodes, but no increase in the usage of BLAS2 routines. To better understand this behavior, we plot in the left figure of Table 4 the distribution of supernode sizes for matrix *twotone* when using the row merge tree (denoted as $twotoneRMT$) and when using the column elimination tree (denoted as $twotoneCOLT$). In both cases, most of the supernodes have size one or two. When using the row merge tree, the number of supernodes of size one is 62295 and the number of supernodes of size two is 6957. When using the column elimination tree, the number of supernodes of size one is 64999 and the number of supernodes of size two is 6337. With both approaches, the largest supernode is of size 36. For the sake of clarity, in the right figure of Table 4 we plot only the supernodes of size 3 to 25. These plots shows that the usage of the row merge tree decreases significantly the number of supernodes of size one while increasing mainly the number of supernodes of size two and three. Since SuperLU uses scalar operations for supernodes of size smaller than 4, this change in the distribution of the supernode sizes is not reflected in the usage of DGEMV routine. We can also observe that the usage of the row merge tree increases significantly the number of supernodes of size 12. However, the impact of this increase on the usage of DGEMV routine is annihilated by the decrease in the number of supernodes of size 8.

6.3 Impact on block triangular form decomposition

For the same subset of matrices used in Tables 2 and 3, we present in Table 5 several characteristics of the blocks on the diagonal obtained after postordering the row merge tree (lines denoted RMT) and after applying the Dulmage Mendelsohn decomposition (lines denoted BTF). Column 3 displays the order of the largest block. Column 4 shows the number of blocks of order 1. Column 5 presents the number of blocks of order between 2 and 10. The last two columns present the number of the remaining blocks on the diagonal and the range of their orders, respectively.

Consider the results obtained by the Dulmage Mendelsohn decomposition. As already presented in the literature for different test matrices [8, 23], for most of the matrices the decomposition leads to a large block on the diagonal. Most of the other blocks are of order 1. For few matrices, like lhr71, this decomposition leads to several non trivial blocks on the diagonal.

Consider the results obtained by postordering the row merge tree. For most of the matrices, a large block is obtained on the diagonal. Except for four matrices, all the other blocks are of order 1.

Matrix	Method	Order Largest Block	Number of blocks of order 1	Number of blocks of order 2-10	Number of other blocks	Order of other blocks
bayer04	RMT	15886	4123	241	0	0
	BTF	13762	6349	19	9	34 - 49
hydr1	RMT	4831	473	2	0	0
	BTF	2370	968	2	3	17 - 1920
lhr71	RMT	68309	1019	330	0	0
	BTF	7663	7038	8	20	1472 - 3664
mahindas	RMT	869	343	16	0	0
	BTF	589	669	0	0	0
orani678	RMT	2133	380	3	1	10
	BTF	1830	699	0	0	0
tols4000	RMT	19	3910	18	0	0
	BTF	18	3982	0	0	0
twotone	RMT	114359	6375	6	0	0
	BTF	105740	12750	380	0	0
west2021	RMT	1708	87	63	1	26
	BTF	1500	521	0	0	0

Table 5: Results comparing the number and the order of the blocks on the diagonal obtained when using the row merge tree (RMT) and when using the Dulmage Mendelsohn decomposition (BTF).

When comparing the results obtained by the row merge tree postordering and by the Dulmage Mendelsohn decomposition in Table 5, as one would expect, Dulmage Mendelsohn decomposition leads to a bigger number of blocks on the diagonal. Generally the largest diagonal block, introduced by postordering the row merge tree, is bigger than the largest diagonal block produced by the Dulmage Mendelsohn decomposition.

6.4 Remarks

The preprocessing phase that UMFPACK uses to identify singletons provides a different approach for the structure prediction of Hall matrices. In this approach, the matrix is permuted such that columns of L that have only the diagonal element nonzero and rows of U that have only the diagonal element nonzero are ordered first. The column elimination tree is used for structure prediction purposes only on the remaining matrix. To compare this approach to our approach, we restricted our attention to the following experiment. Consider S the submatrix obtained by removing from A the columns of L and the rows of U that have only the diagonal element nonzero. We analyzed the structure prediction of submatrix S using the column elimination and then using the row merge tree. When analyzing the decomposition in a block triangular form, for most of the matrices the row merge tree identifies a small number of blocks on the diagonal. One of the exceptions is matrix *twotone*, which has 381 blocks on the diagonal. When analyzing the estimation of upper bounds for the structure of L and of U, the decrease obtained by the usage of the row merge tree is generally not significant. When analyzing the supernodes, the usage of the row merge tree tends to create slightly smaller number of supernodes. But there are cases when the usage of the column elimination tree leads to a slightly smaller number of supernodes. These experiments make us believe that the usage of the row merge tree is not motivated in this case.

A more thorough analysis is necessary for an overall comparison of the usage of the row merge tree on the matrix A with the approach used in UMFPACK. But this is beyond the scope of the paper.

We also studied the row merge tree for matrices that satisfy the strong Hall property and we noticed that its usage does not introduce significant extra time. Moreover its structure is equal to the structure of the column elimination tree for these matrices.

7 Conclusions

In this paper we have studied properties of the row merge tree and its role in the Gaussian elimination with partial pivoting of matrices that satisfy only the Hall property. We have presented new theoretical results that relate the nonzero structure of rows in the lower triangular part of the row merge matrix to branches of the row merge tree, and the nonzero structure of columns in the upper triangular part of the row merge matrix to subtrees of the row merge tree, respectively. These results allowed us to develop an efficient algorithm for building the row merge tree, that uses as input solely the structure of A. We have also analyzed the practical interest of the row merge tree by studying experimentally its usage as an alternative to the usage of the column elimination tree. We have used a set of 19 matrices that satisfy only the Hall property. We have observed that only for a limited number of matrices (8 matrices in our test set), the use of the row merge tree provides a better alternative than the column elimination tree. For these matrices, it can better predict similarities between columns and thus a postordering on this tree can decrease the number of supernodes. It can be used to determine the counts of the row merge graph efficiently, thus decreasing the memory requirements.

The structure of the column elimination tree is independent of any row permutation. The row merge tree provides a more precise estimation of the structure of L and U if the matrix has a zero free diagonal. Hence its usage introduces a supplementary time to permute the matrix to a zero free diagonal form, and this can present one major drawback. However, there are cases when it is interesting for different reasons to permute the matrix to a zero free diagonal form. For example, a very used approach consists in permuting large entries on the diagonal [9] to decrease pivoting during the numerical factorization.

Other studies of the row merge tree structure can be found in [20] and [22]. Liu [20] uses the row merge tree to propose a computational sequence of steps for performing Givens rotations. The assumption taken in that paper is that the Cholesky factor of A^TA has the same structure as the orthogonal factor R, condition satisfied when the matrix has the strong Hall property (which is a difference from the method presented in this paper, in that we assume only the Hall property). Another difference is related to the structure of the tree. The leaves of the row merge tree in [20] correspond to rows in the matrix, while the nodes correspond to columns in the matrix. The tree structure studied here considers only the nodes corresponding to columns in the matrix.

Oliveira [22] presented a different method for building row merge trees for matrices satisfying only the Hall property. The approach described in that paper is as follows. First the column elimination tree is computed. Then the number of nonzeros in each column of $L^\times$ is computed, thus identifying columns with only zero elements. Then two post-processing steps are applied on the column elimination tree in order to delete several edges, thus rendering this tree into the row merge tree. In this paper we take a different approach in that we compute the structure of the row merge tree in one step, using as input the structure of the original matrix A.

The row merge tree structure discussed in this paper can find usage in several solvers. Solvers like the sequential and shared memory versions of SuperLU, can consider the use of the row merge tree instead of the column elimination tree for postordering and guiding the parallel execution. Solvers like S+, S* [25] use the structure of $L^{\times}$, $U^{\times}$ during the numerical factorization, thus allowing operations on possibly zero entries. These solvers already use the row merge tree, but its structure is computed after computing the row merge graph. No postordering step is applied. Computing first the row merge tree, applying a postordering step, and finally computing the structure of the row merge graph can lead to increase in BLAS usage and decrease in time for the symbolic factorization step of this solver.

8 Acknowledgments

The authors thank Xiaoye S. Li for providing the code to compute the column and row counts of the upper bounds on structure of L and U using the column elimination tree; Alex Pothen for providing the code to compute the block triangular form of a sparse matrix; John R. Gilbert for making available the code to compute the column elimination tree, for suggesting the presentation of a fill path as a tree structure used in Section 2, and for helpful discussions on this topic. The authors would like to thank the anonymous reviewers for their helpful comments and suggestions to improve the paper.

References

[1] H. Avron, G. Shklarski, and S. Toledo. Parallel unsymmetric-pattern multifrontal sparse lu with column preordering. *Submitted for publication in ACM Trans. Math. Software*, 2004.

[2] R. A. Brualdi and B. A. Shader. Strong Hall matrices. *SIAM J. Matrix Anal. Appl.*, 15:359–365, 1994.

[3] T. A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Software*, 30(2):165–195, 2004.

[4] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. Liu. A Supernodal Approach to Sparse Partial Pivoting. *SIAM J. Matrix Anal. Appl.*, 20(3):720–755, 1999.

[5] J. W. Demmel, J. R. Gilbert, and X. S. Li. An Asynchronous Parallel Supernodal Algorithm for Sparse Gaussian Elimination. *SIAM J. Matrix Anal. Appl.*, 20(4):915–952, 1999.

[6] I. S. Duff. Algorithm 575. Permutations for a zero-free diagonal. *ACM Trans. Math. Software*, 7:387–390, 1981.

[7] I. S. Duff. On Algorithms for Obtaining a Maximum Transversal. *ACM Trans. Math. Software*, 7:315–330, 1981.

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

[9] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal. Appl.*, 22(4):973–996, 2001.

[10] 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(2):137–147, June 1978.

[11] A. George and E. Ng. Symbolic factorization for sparse gaussian elimination with partial pivoting. *SIAM J. Sci. Statist. Comput.*, 8(6):877–898, 1987.

[12] A. George and E. G. Ng. Parallel Sparse Gaussian Elimination with Partial Pivoting. *Annals of Operations Research*, 22:219–240, 1990.

[13] J. R. Gilbert. An efficient parallel sparse partial pivoting algorithm. Technical Report 88/45052-1, Christian Michelsen Institute, 1988.

[14] J. R. Gilbert, X. S. Li, E. G. Ng, and B. W. Peyton. Computing Row and Column Counts for Sparse QR and LU Factorization. *BIT*, 41(4):693–711, 2001.

[15] J. R. Gilbert and E. G. Ng. Predicting Structure in Nonsymmetric Sparse Matrix Factorizations. In A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, pages 107–139. Springer Verlag, 1994.

[16] L. Grigori. *Structure Prediction and Parallel Algorithms for LU Facorization of Sparse Matrices.* PhD thesis, University Henri Poincare, France, 2001.

[17] L. Grigori, J. Gilbert, and M. Cosnard. Structure Prediction for Sparse Gaussian Elimination with Partial Pivoting. *Submitted for publication*, 2005.

[18] J. W. H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11:141–153, 1985.

[19] J. W. H. Liu. A Compact Row Storage Scheme for Cholesky Factors Using Elimination Trees. *ACM Trans. Math. Software*, 12(2):127–148, June 1986.

[20] J. W. H. Liu. On General Row Merging Schemes for Sparse Givens Transformations. *SIAM J. Sci. Statist. Comput.*, 7(4), October 1986.

[21] J. W. H. Liu. The Role of Elimination Trees in Sparse Factorization. *SIAM J. Matrix Annal. Appl.*, 11(1):134–172, 1990.

[22] S. Oliveira. Exact Prediction of QR fill-in by row-merge trees. *SIAM J. Sci. Comput.*, 22(6):1962–1973, 2001.

[23] A. Pothen and C.-J. Fan. Computing the Block Triangular Form of a Sparse Matrix. *ACM Trans. Math. Software*, 16(4):303–324, 1990.

[24] K. Shen, X. Jiao, and T. Yang. Elimination Forest Guided 2D Sparse LU Factorization. *ACM Symp. on Parallel Architectures and Algorithms (SPAA'98)*, pages 5–15, 1998.

[25] K. Shen, T. Yang, and X. Jiao. S+: Efficient 2d Sparse LU Factorization on Parallel Machines. *SIAM J. Matrix Anal. Appl.*, 2000.

[26] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22:215–225, 1975.