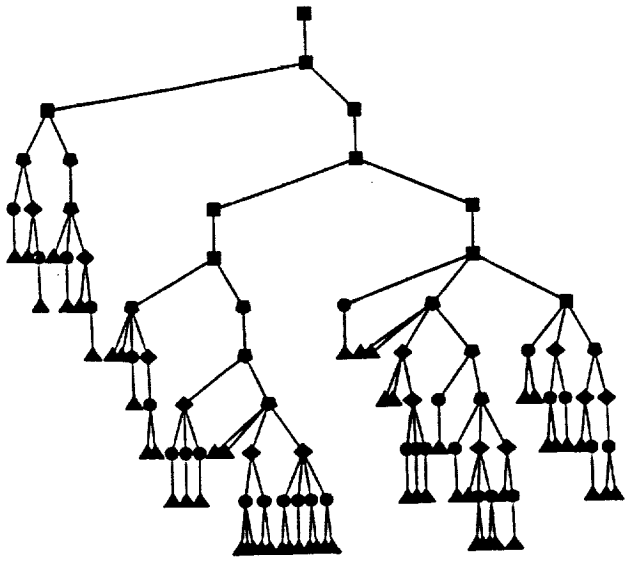
Research Institute for Advanced Computer Science
NASA Ames Research Center

Optimal Parallel Solution of Sparse Triangular Systems

Fernando L. Alvarado Robert Schreiber

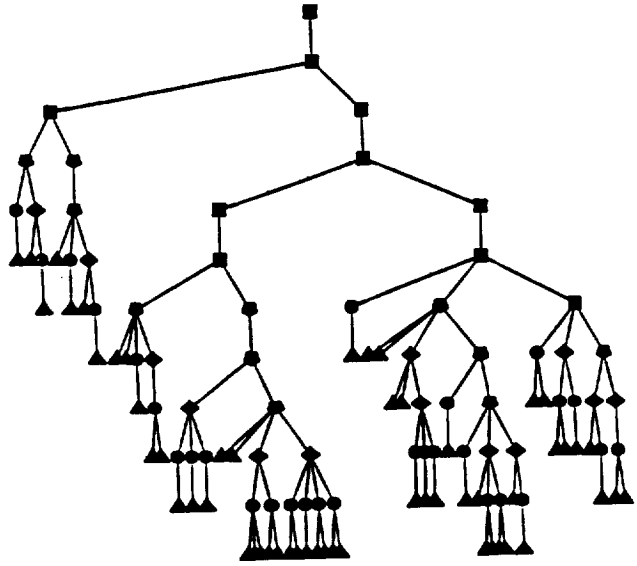


RIACS Technical Report 90.36 September, 1990

Submitted: SIAM Journal on Scientific and Statistical Computing

Optimal Parallel Solution of Sparse Triangular Systems

Fernando L. Alvarado Robert Schreiber



The Research Institute of Advanced Computer Science is operated by Universities Space Research Association, The American City Building, Suite 311, Columbia, MD 244, (301)730-2656

Work reported herein was supported by the NAS Systems Division of NASA and DARPA via Cooperative Agreement NCC 2-387 between NASA and the University Space Research Association (USRA). Work was performed at the Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA 94035.

Optimal Parallel Solution of Sparse Triangular Systems

Fernando L. Alvarado*
The University of Wisconsin—Madison
Madison, Wisconsin 53706, USA

Robert Schreiber†
RIACS
MS 230-5, NASA Ames Research Center
Moffett Field, California 94035

June 8, 1990

Abstract

This paper describes a method for the parallel solution of triangular sets of equations, appropriate when there are many right-hand sides. By preprocessing, the method can reduce the number of parallel steps required to solve $Lx = b$ compared to parallel forward or backsolve. Applications are to iterative solvers with triangular preconditioners, to structural analysis, or to power systems applications, where there may be many right-hand sides (not all available *a priori*).

The inverse of L is represented as a product of sparse triangular factors. The problem considered in this paper is to find a factored representation of this inverse of L with the smallest number of factors

*This work was supported in part under NSF Contracts ECS-8822654 and ECS-8907391.

†This work was supported by the NAS Systems Division and/or DARPA via Cooperative Agreement NCC 2-387 between NASA and the University Space Research Association (USRA). Work was performed at the Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA 94035 and at CERFACS, 42 Ave. Gustave – Coriolis, 31057 Toulouse Cedex, France. The assistance of CERFACS and its staff during this author's visit in August 1989 are gratefully acknowledged.

(or partitions), subject to the requirement that no new nonzero elements be created in the formation of these inverse factors. A method from an earlier reference is shown to solve this problem. This paper improves upon this method by contructing a permutation of the rows and columns of L that preserves triangularity and allows for the best possible such partition.

A number of practical examples and algorithmic details are presented. The parallelism attainable is illustrated by means of elimination trees and cliques trees.
Keywords: Sparsity, sparse matrices, numerical linear algebra, triangular matrices, matrix partitioning, parallel computation.

1 Introduction

This paper considers the problem of solving a nonsingular lower triangular set of sparse equations $Lx = b$ in a parallel environment. (All our results extend in a trivial way to upper triangular systems, a fact we shall not mention again.) We consider the case where the problem must be solved for multiple right hand side vectors b, and these vectors are not available all at once. By preprocessing, we reduce the number of parallel steps required.

Important applications where multiple right-hand-sides arise include finite element applications, preconditioned iterative solvers for linear systems, solution of initial value problems by implicit methods, and variants of Newton's method for the solution of nonlinear equations. Often, L is a triangular factor computed by LU decomposition of a sparse matrix. In this case, L is a perfect elimination matrix (its graph is chordal). Our results do not require this. Thus, L may arise from an incomplete factorization or any other process.

There are two possible approaches to the parallel solution of triangular systems of equations. The usual approach is to exploit whatever parallelism is available in the usual substitution algorithm [4, 10]. The second, which requires preprocessing, works with some representation of L^{-1}. In sequential sparse matrix computation, substitution is universally favored because it retains the sparsity of the problem [7]. If L is dense, its inverse L^{-1} is also dense. If L is sparse, its inverse is usually much denser than L itself [9].

Here we consider a factorization

$$L^{-1} = \prod_{k=1}^{m} Q_k \tag{1}$$

with sparse factors. Such a factorization is possible in which the factors have no more nonzeros than L [2, 3, 9]. The chief advantage of a factorization of L^{-1} is that all the necessary multiplications for the computation of $Q_k x$ can be performed concurrently. Thus, it is possible to take advantage of more parallelism in the solution of these equations. The necessary additions can be done in at most $\log_2 m$ operations, where m is the largest number of nonzeros in a row of Q_k.

The remainder of this introduction reviews the use of partitioned inverses of L. Any triangular matrix L can be expressed as a product of elementary matrices:

$$L = L_1 L_2 \cdots L_{n-1} \tag{2}$$

where

$$\begin{aligned} (L_k)_{ik} &= L_{ik} \quad \text{for} \quad k \le i \le n, \\ (L_k)_{jj} &\equiv 1 \quad \text{for} \quad j \ne k, \\ (L_k)_{ij} &\equiv 0 \quad \text{otherwise}. \end{aligned}$$

The matrices L_k are known as elementary lower triangular matrices. They can be grouped into several factors,

$$L = \prod_{k=1}^{m} P_k \tag{3}$$

where

$$P_k = L_{e_{k-1}+1} L_{e_{k-1}+2} \cdots L_{e_k} \tag{4}$$

and

$$0 = e_0 < e_1 < \cdots < e_m = n - 1. \tag{5}$$

Here $\{e_k\}_{k=0}^{m}$ is a monotonically increasing integer sequence. The factor P_k is lower triangular and is zero below its diagonal in all columns except columns $e_{k-1}+1$ through e_k, where it is identical to L. Consider, for example:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 3 & 2 & 3 & 0 \\ 4 & 0 & 1 & 1 \end{bmatrix}$$

By choosing $e_0 = 0; \quad e_1 = 2; \quad e_2 = 3$, L can be partitioned as follows:

$$L = L_1 L_2 L_3 = (L_1 L_2) L_3 = P_1 P_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 3 & 2 & 1 & 0 \\ 4 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

The solution of the partitioned problem proceeds as follows. From (3) it follows that

$$x = L^{-1} b = \prod_{k=m}^{1} P_k^{-1} b. \tag{6}$$

Thus, one may compute x as follows:

```
x = b;
for k = 1 to m do
    x = P_k^{-1} x;
    od;
```

In computing the matrix-vector products, we may exploit parallelism fully, computing the results in $\lceil \log_2 \text{max-row}(P_k^{-1}) \rceil$ time with $\eta(P_k^{-1})$ processors. Here $\eta(X)$ denotes the number of nonzeroes in X and $\text{max-row}(X) = \max_i$ (number of nonzeroes in row i of X) $= \max_i(\eta(e_i^T X))$.

Our viewpoint, which is the use of a partitioned inverse of L, is computationally quite similar to frontal methods for factorization [8] and to methods using supernodes [5].

2 Problem Definition

This section defines some terminology and the two problems that are addressed in this paper.

Definition 1 *X is* invertible in place *if $x_{ij} \neq 0 \Leftrightarrow (X^{-1})_{ij} \neq 0$.*

The elementary lower triangular matrices are invertible in place. There therefore always is at least one partition (3) of L with factors that invert in place. A partition in which the factors P_k are invertible in place is called a "no-fill" partition. The objective here is to find a no-fill partition with the smallest possible number of factors.

Problem 1 *Determine m and $e_1, \cdots, e_{m-1}$ such that each P_k, $1 \leq k \leq m$, is invertible in place and m is minimum. Denote this minimum m by* min-fact(L).

Definition 2 *A solution to Problem 1 is a* minimum no-fill partition *of L.*

Problem 2 *Determine a permutation matrix Π such that $L_\Pi \equiv \Pi L \Pi^T$ remains lower triangular and* min-fact(L_Π) *is minimized.*

2.1 Graph Theory Concepts

Let $G(L)$ be a digraph with vertices $V = \{1, 2, \ldots, n\}$ and directed edges $E = E(L) \equiv \{(i,j) \mid L_{ij} \neq 0\}$. Think of the edge (i,j) as an arrow going from vertex j to vertex i. If L is lower triangular, then for all edges (i,j) we have $i > j$. $G(L)$ is therefore an acyclic digraph, or DAG.

Definition 3 *The* indegree *of a vertex i is the number of vertices j such that $(i,j) \in E$.*

The indegree of vertex i is the number of nonzeros in the i^{th} row of L.

Definition 4 *Let $G = G(L)$. For $j \in V$, $madj(j)$ is the set of higher numbered neighbors of j, i.e. the set $\{i > j \mid L_{ij} \neq 0\}$.*

The set $madj(j)$ is the set of rows that are nonzero in the j^{th} column of L.

Definition 5 *For $(i,j) \in E$ we say that j is a predecessor of i and i is a successor of j.*

In anticipation of forthcoming theorems, we need to establish two lemmas.

Lemma 1 *If L is a nonsingular lower triangular matrix, then (i,j) is an edge of $G(L^{-1})$ iff there is a path from j to i in $G(L)$.*

Proof: For all $1 \leq i \leq n$, $(i,i) \in E(L) \cap E(L^{-1})$, since both are nonsingular and triangular. We induct on $i - j$. For $i > j$

$$\begin{aligned} 0 &= (LL^{-1})_{ij} \\ &= \sum_{p=j}^{i} L_{ip} L_{pj}^{-1} \\ &= \left(\sum_{p=j}^{i-1} L_{ip} L_{pj}^{-1} \right) + L_{ii} L_{ij}^{-1} \end{aligned}$$

The second term above is nonzero iff $L_{ij}^{-1} \neq 0$. The first term is nonzero iff at least one product $L_{ip} L_{pj}^{-1} \neq 0$, $j \leq p \leq i-1$. By the inductive hypothesis (since for any such p, $p - j < i - j$) this holds iff there is an edge (i,p) and a path from j to p. Together, these constitute the required $j \rightarrow i$ path. □

Definition 6 *The digraph $G = (V,E)$ is closed if $(i,k) \in E$ and $(k,j) \in E \Rightarrow (i,j) \in E$.*

Definition 7 *Let $G = (V, E)$ be a digraph associated with a triangular matrix L. Given a subset S of V, define the* column subgraph *$G_S = (V, E_S)$ (where $E_S \equiv \{(i,j) \in E | j \in S\}$) as the graph of the lower triangular matrix obtained by zeroing all columns of L not in S.*

Theorem 2 *Let a partition (5) and corresponding factorization (3) be given. The factors P_k are invertible in place iff each column subgraph $G(P_k) = G_{\{e_{k-1}+1,\ldots,e_k\}}$ is closed.*

Proof: By Lemma 1 $(P_k^{-1})_{ij} \neq 0$ iff there is a $j \rightarrow i$ path in $G(P_k)$. The following are therefore equivalent:

- P_k is invertible in place
- $(P_k^{-1})_{ij} \neq 0 \Rightarrow (P_k)_{ij} \neq 0$
- for every $j \rightarrow i$ path, $(P_k)_{ij} \neq 0$
- G(P_k) is closed.

□

Lemma 3 *Let L_1 and L_2 be lower triangular. Then $(i,j) \in E(L_1 L_2)$ iff there is some k, with $j \leq k \leq i$ such that $(i,k) \in E(L_1)$ and $(k,j) \in E(L_2)$.*

Proof: For all $i > j$, $(L_1 L_2)_{ij} = \sum_{k=j}^{i} (L_1)_{ik}(L_2)_{kj}$. See Figure 1.

2.2 Partitioning the Inverse

Consider Problem 1, that of representing L^{-1} by partitioning its factorization with the smallest possible number of factors that invert in place. The following algorithm was proposed by Alvarado, Yu and Betancourt (who call it PA2) [3]:

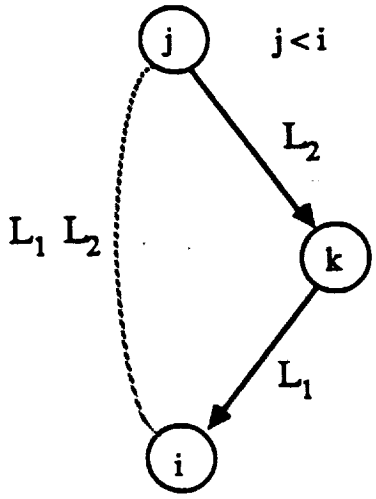


Figure 1: Proof of Lemma 3.

Algorithm P1:
Input: $L = L_1 L_2 \cdots L_{n-1}$
Output: A partition of L.

$i \leftarrow 1$; $k \leftarrow 1$;
while $(i < n-1)$ **do**
  let r be the largest integer greater than i such that $L_1 \cdots L_r$
    is invertible in place;
  $P_k \leftarrow L_i \cdots L_r$;
  $k \leftarrow k+1$; $i \leftarrow r+1$;
  od

Alvarado, Yu, and Betancourt did not mention the issue of minimality. Here we show that Algorithm P1 determines a minimum no-fill partition.

Lemma 4 *If $L_1 \cdots L_r$ is invertible in place, then $L_2 \cdots L_r$ is, too.*

Proof: Obvious.

Theorem 5 *Algorithm P1 produces a minimum partition (it solves Problem 1).*

Proof: Suppose Algorithm P1 produces a partition $L = P_1 \cdots P_m$. Clearly there does not exist a no-fill partition with e_1 any larger than that produced by Algorithm P1.

Now we show by induction on n that there is no better partition. Let $L = \hat{P}_1 \cdots \hat{P}_{m'}$ be a different no-fill partition. Suppose $P_1 = \hat{P}_1$. By the induction hypothesis, Algorithm P1 when applied to $P_2 \cdots P_m$, produces a minimum no-fill partition, i.e. $m-1$ factors is least possible. Thus, $m' \geq m$.

On the other hand, perhaps $\hat{P}_1 = L_1 \cdots L_{e_1'}$ with $e_1' < e_1$, i.e. $\hat{P}_1$ has *fewer* nonzero columns than does P_1. The matrix $\hat{Q} \equiv \hat{P}_2 \cdots \hat{P}_{m'}$ has a no-fill partition with $(m'-1)$ factors. By the previous lemma, we may remove the leftmost elementary lower triangular factor of $\hat{Q}$ and still have an $m'-1$ factor no-fill partition. Continuing in this way we find such a partition of Q. But by the inductive hypothesis, any no-fill partition of Q has at least $m-1$ factors. Thus, we again see that $m' \geq m$. □

Minimum partitions are not unique. For example,

$$L = \begin{bmatrix} \mathtt{x} & & & \\ \mathtt{x} & \mathtt{x} & & \\ \mathtt{x} & \mathtt{x} & \mathtt{x} & \\ 0 & 0 & \mathtt{x} & \mathtt{x} \end{bmatrix} \tag{7}$$

has minimum partitions:

$$L = (L_1 L_2 L_3) L_4$$

and

$$L = (L_1 L_2)(L_3 L_4)\,.$$

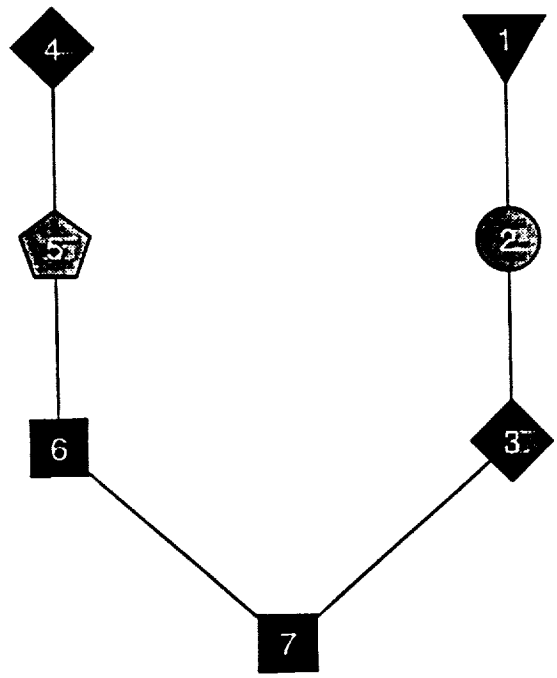


Figure 2: Illustration of the minimum partition for a graph without reordering. Five factors are required.

2.3 Triangular Permutation

Consider the graph illustrated in Figure 2 for which L has a minimum partition:

$$L = (L_1)(L_2)(L_3)(L_4)(L_5)(L_6L_7)$$

This partition has six factors. It is possible to symmetrically permute the rows and columns of L such that L remains a lower triangular and $G(L)$ is as illustrated in Figure 3. A minimum partition of L for this new graph is:

$$L = (L_1L_2)(L_3L_4)(L_5L_6L_7)$$

This permuted partitioned matrix has only three factors.

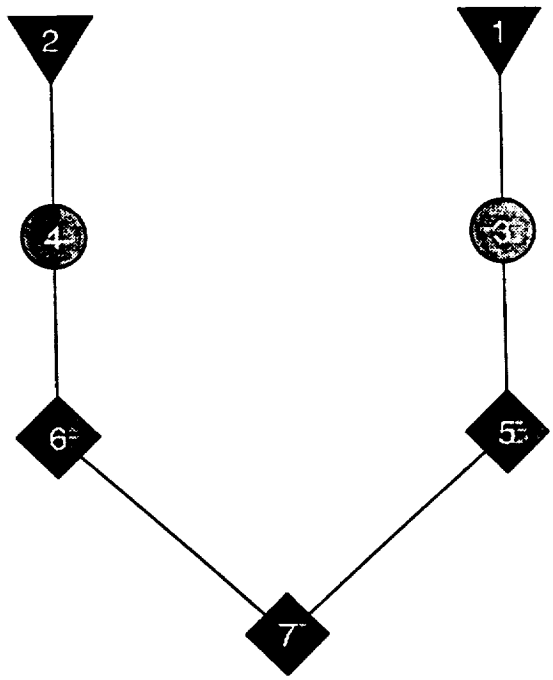


Figure 3: Illustration of the minimum partition for a re-ordered graph. Only three factors are required.

Definition 8 *A* triangular ordering *of an acyclic digraph is a numbering of vertices in which* $(i,j) \in E$ *implies* $i > j$.

Every acyclic digraph has triangular orderings. The digraph of every triangular matrix is acyclic; moreover, its conventional ordering is triangular. If $G(L)$ is re-ordered so that vertex i is numbered $\nu(i)$ and the new ordering is triangular, then the symmetric permutation of rows and columns that moves row i to row $\nu(i)$ leaves L triangular.

2.4 Level of a Vertex

Let $G = (V, E)$ be a DAG. Define level$(i), i \in V$ as follows [6]:

1. If indegree$(i) = 0$, then level$(i) = 0$.
2. Otherwise, level$(i) = 1 +$ level(i'), where i' is the vertex corresponding to i in the subgraph of G obtained by deleting all vertices at level 0 and their outgoing edges.

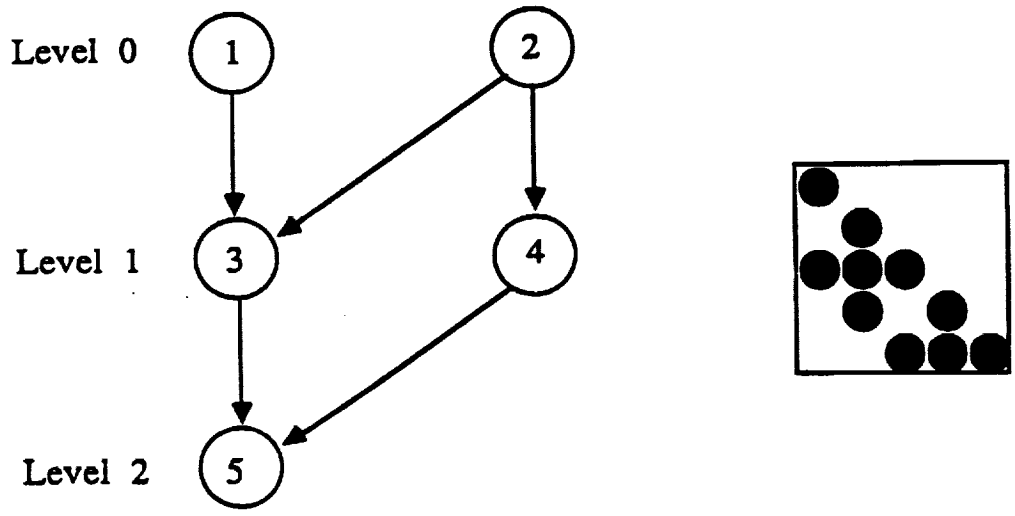


Figure 4: Illustration of the concept of Level.

Figure 4 illustrates the concept of level. In general, level(i) is the length of the longest path that ends at i. The computation of levels is straightforward, requiring $O(\eta(L))$ time.

3 Best Partitioning with Re-ordering

This section describes two new algorithms for solving Problem 2. Each of them finds a triangular ordering of a DAG G such that the re-ordered graph has a minimum no-fill partition with the smallest possible number of factors. The first algorithm (RP1) is a fairly straightforward "greedy" algorithm. The second algorithm, RP2, is a faster but more complicated implementation of the same idea.

Algorithm RP1 (Re-order, Permute 1):

Input: A directed, closed acyclic digraph $G(L)$.

Output: A permutation $\nu : V \longrightarrow \{1, \ldots, n\}$ *and* a partition of L.

Compute level(v) for all $v \in V$;
max-level $\leftarrow \max_{v \in V}(\text{level}(v))$;
$i \leftarrow 1$; $k \leftarrow 1$; $e_0 \leftarrow 0$;
while $i < n$ **do**
  $S_k \leftarrow \emptyset$; $e_k \leftarrow i$;
  $\ell \leftarrow min\{j \mid$ there is an unnumbered vertex at level $j\}$;
  repeat
    for every vertex v at level ℓ **do**
      if (([Condition 1] v is unnumbered) &&
        ([Condition 2] Every predecessor of v has been numbered) &&
        ([Condition 3] Every successor of v is a successor of all
          $u \in S_k$ such that u is a predecessor of v)) **then**
        $\nu(v) \leftarrow i$; $i \leftarrow i + 1$;
        $S_k \leftarrow S_k \cup \{v\}$; $e_k \leftarrow e_k + 1$;
        fi
      od
    $\ell \leftarrow \ell + 1$;
    until $\ell >$ max-level or no vertices at level $\ell - 1$ were included in S_k;
  $P_k \leftarrow L_{e_{k-1}} \cdots L_{e_k}$; $k \leftarrow k + 1$;
  od

The algorithm works by finding a partition $V = \cup_{k=1}^{m} S_k$ for which the column subgraphs G_{S_k} are closed. Moreover, S_1 is a source node in the quotient graph, i.e. there are no edges directed into S_1. If S_1 and its out-edges are removed, then S_2 is a source node, etc. Thus, we are carrying out a partitioning of G as well as of L. We shall call the subsets S_k in this partition *factors* in analogy with their corresponding factors P_k of L.

Proposition 1 *The factor S_1 chosen by RP1 is the largest possible*, i.e., S_1 *includes all of* level 0*; all allowable nodes from* level 1*; all allowable nodes from* level 2; etc. *This is also true of $S_j, j > 1$, which is the largest possible*

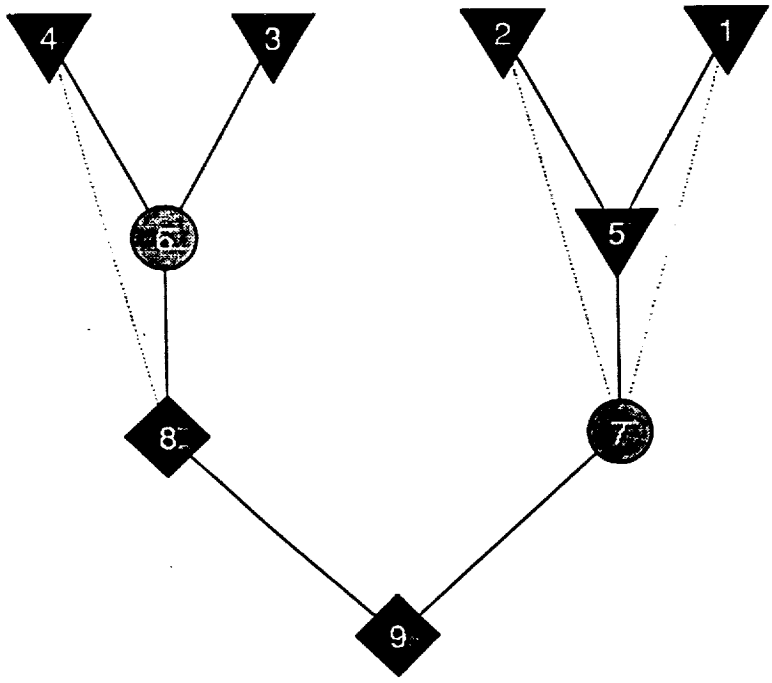


Figure 5: Illustration of Condition 3. Successors of predecessors of a node must be successors of the node itself. Node 5 can be included in the same factor as nodes 1 and 2, but node 6 cannot be included with 3 and 4. Factors are denoted by shapes and shading.

first factor of the graph obtained by deleting all verices and incident edges of $S_1, \ldots, S_{j-1}$.

This proposition is self-evident. We illustrate it with the example in Figures 5 and 6. These figures illustrate the elimination tree [13] of a matrix, and illustrate all successors of each node as well as the next element in the tree. Different node shapes are used to denote different factors. In Figure 5 it is possible to combine nodes 1, 2 and 5 into the same factor according to Condition 3: all the successors of the predecessors of 5 are successors of 5 itself. It is *not* possible, however, to combine node 6 into the same factors as nodes 3 and 4 because this rule is violated. Figure 6 illustrates a case where it would be possible to merge node 5 into the same factor as nodes 2 and 3, but not all the predecessors of 5 are numbered when node 5 becomes eligible (node 4 is not yet numbered when node 5 is considered). This violates Condition 2. Thus, node 5 must be in a different factor.

Theorem 6 *Procedure RP1 solves Problem 2.*

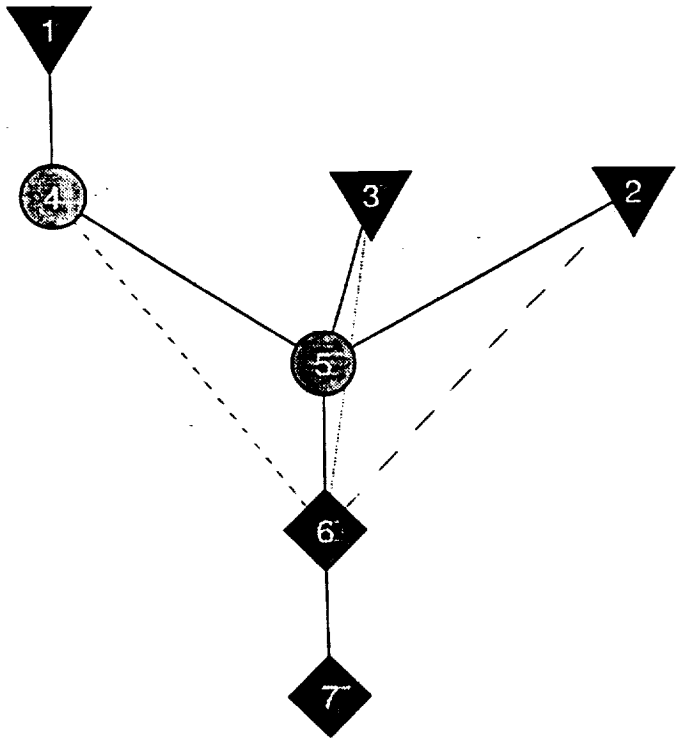


Figure 6: Illustration of Condition 2. A node must be excluded from a factor because some of its predecessors are not numbered. One predecessor of node 5 (node 4) has not been numbered by the time node 5 becomes eligible to join the factor containing nodes 2 and 3.

Proof: By induction on n. Consider any permutation Γ and partition $L_\Gamma = \Gamma L \Gamma^T = \prod_{k=1}^{m} P_k$ with P_k defined by (4) and (5), and with P_k invertible in place. We claim that the partition point e_1 is no larger than that produced by RP1. For it follows from the construction of P_1 that any vertex of $G(L)$ not included in S_1 by RP1 either has an unnumbered predecessor, or else its inclusion renders G_{S_1} not closed. Hence the subset S_1 selected by RP1 is maximal. Now, by the inductive hypothesis, RP1 partitions $G \setminus S_1$, using $m-1$ invertible-in-place factors. Moreover, by Lemma 4, a graph obtained by adding source vertices to $G \setminus S_1$ requires at least $m-1$ factors in any optimal partition. Thus, if we take a subset $\hat{S}_1$ of S_1 as the first factor, we can achieve nothing better. □

We would like to insure that running time is bounded by a small constant multiple of $\eta(L)$. But the running time of Algorithm RP1 is large in some cases. Consider a dense lower triangular matrix of order n. RP1 takes $O(n^3)$ time in this case, since the cost of checking whether all successors of vertex j are also successors of its predecessors is $O\left(j(n-j)\right)$.

We now introduce two new data structures in order to improve the efficiency of RP1. We can improve the performance of RP1 (to $O(\eta(L))$ for a dense matrix) by insuring that a vertex is not examined for possible inclusion in S_k until all of its predecessors have been numbered. To do so, for every vertex we count the number of its unnumbered predecessors. Initially, this is its indegree. When the count reaches zero we can consider the vertex for inclusion in a factor.

Second, we can avoid much of the work associated with the checking at Condition 3 of RP1. Let u and v be numbered vertices both of which have been included in the current factor S_k. Assume that v is a successor of u. Then we must have that $\text{madj}(v) \subseteq \text{madj}(u)$, otherwise v would have failed the test at Condition 3. Thus, we need not consider vertex u when applying the requirements of Condition 3 to a vertex that is also a successor of v. We shall make use of this in the faster implementation (RP2, below) by keeping track of the set of predecessors of each vertex that may need to be examined in checking Condition 3. In the situations above, when v is included in S_k we remove u from the predecessor sets of v's successors, thus avoiding the unnecessary checking.

Algorithm RP2 (Re-order, Permute 2):

Input: A directed, closed acyclic digraph $G(L)$.

Output: A permutation *and* a partition of L.

```
forall v ∈ V do
    pred(v) ← {u | L_vu ≠ 0};
    count(v) ← indegree(v);
    Compute level(v);
    od
max-level ← max_{v∈V}(level(v));
i ← 1;   k ← 1;   e_0 ← 0;
E ← {v ∈ V | count(v) = 0};
while i < n do
    S_k ← ∅;   e_k ← i;
    ℓ ← min{j | there is an unnumbered vertex at level j};
    repeat
        for every vertex v ∈ E at level ℓ do
            if ( [Condition 3'] Every successor of v is a successor of all
                        u ∈ pred(v) ) then
                ν(v) ← i;   i ← i + 1;
                S_k ← S_k ∪ {v};   e_k ← e_k + 1;
                for every successor w of v do
                    pred(w) ← pred(w) \ pred(v);
                    count(w) ← count(w) − 1;
                    if count(w) = 0 then E ← E ∪ {w};
                    od
                fi
            od
        ℓ ← ℓ + 1;
        until ℓ > max-level or no vertices at level ℓ − 1 were included in S_k;
    P_k ← L_{e_{k−1}} ··· L_{e_k};   k ← k + 1;
    od
```

The computation of pred(v) is straightforward, requiring $O(\eta(L))$ time. Clearly, the innermost loop of Algorithm RP2 is executed $\eta(L)$ times. For

we have that the sum of count(v) over all vertices v is just $\eta(L)$, and this sum decreases by one for every execution of this innermost loop. The other statements in the scope of the then clause are executed no more than n times. It is still possible that testing Condition 3' will be costly. Indeed, one can construct examples for which the running time is greater than $\eta(L)$, but in practice this is unlikely to happen.[1]

4 Examples

This section illustrates several examples of the performance of the proposed algorithms for matrix partitioning, and compares the behavior of the proposed algorithm with the performance of previous algorithms. Two tables of comparative data are presented. Table 1 uses five power system matrices ranging is size from 118 to 1993. Table 2 gives results for matrices arising from five-point finite difference discretizations.

In each case, the original coefficient matrix is first ordered and fills are added to make it a perfect elimination matrix. We need to distinguish this first fill-reducing ordering of A from the re-ordering of L found by RP1 and RP2. We call the ordering of A the primary ordering. Three primary ordering procedures are used: the minimum degree algorithm [14], the multiple minimum degree (MMD) algorithm [11], and the minimum level, minimum degree (MLMD) algorithm [6].

For each matrix and primary ordering algorithm, two partitioning methods are compared: Algorithm P1, which simply partitions L optimally without re-ordering it, and Algorithm RP1 which re-orders the matrix and generates an optimal partition. In most cases, Algorithm RP1 gives a smaller number of factors than PA1, while in a few cases both algorithms give the same number of factors.

The main observation justified by these data is that RP1 reduces the number of factors at no expense in added fills. Its effect is most dramatic if the underlying primary ordering is the minimum degree algorithm. On the other hand, the best results are obtained when the MLMD algorithm is used for the primary ordering, even though the relative improvement

[1] Consider a graph with $3k$ vertices arranged in 3 levels of k each. All vertices at level ℓ are adjacent to all at level $\ell+1$, for $\ell = 1, 2$. Running time is $O(\eta(L)^{3/2})$ again.

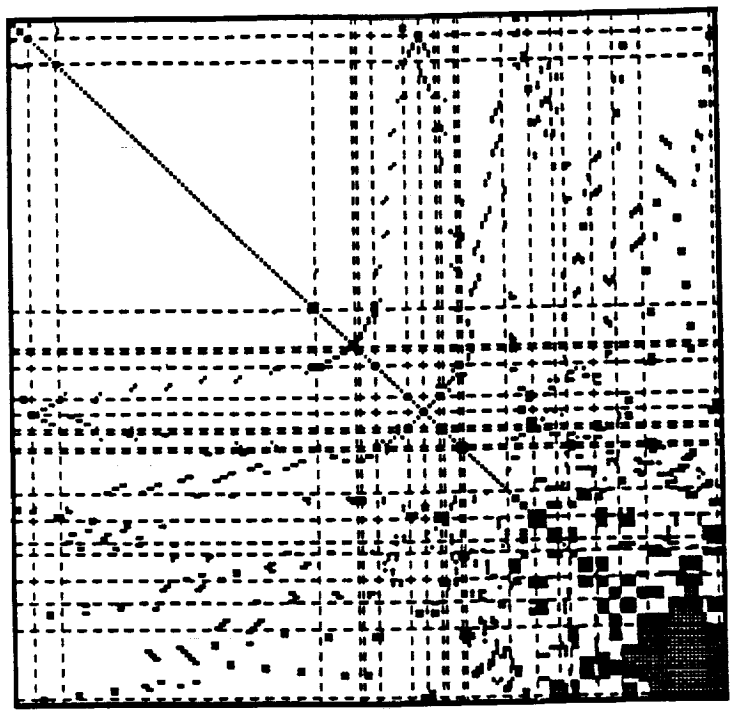


Figure 7: Five-point finite difference matrix for 10 by 10 grid. Matrix ordered by the minimum degree algorithm, then partitioned by P1. Twenty factors result.

attainable by Algorithm RP1 over Algorithm P1 is small. The results for the MMD algorithm fall somewhere between minimum degree and MLMD. The reduction in the number of factors achieved by RP1 in comparison with P1 is quite dramatic when MMD is the primary ordering.

Figures 7 through 10 provide a different illustration of the effect of all four algorithmic combinations considered. All four figures use a 10 by 10 finite difference grid. Figure 7 illustrates the original matrix ordered by the minimum degree method, fills added, and its partition obtained using Algorithm P1. Twenty factors can be seen. Figure 8 illustrates the same matrix after re-ordering it by RP1. Although the matrix has been re-ordered, its topology is identical with that of Figure 7, but only twelve factors are required. Figure 7 illustrates the original matrix ordered by the MLMD algorithm after fills are added (its DAG is chordal). The topology of L is different from the one in the previous two figures. It is, in fact, a little denser. It can be partitioned with Algorithm P1 using only nine factors. Finally, if the original matrix is ordered with the MLMD algorithm and then re-ordered and partitioned with RP1, the result is a matrix with the

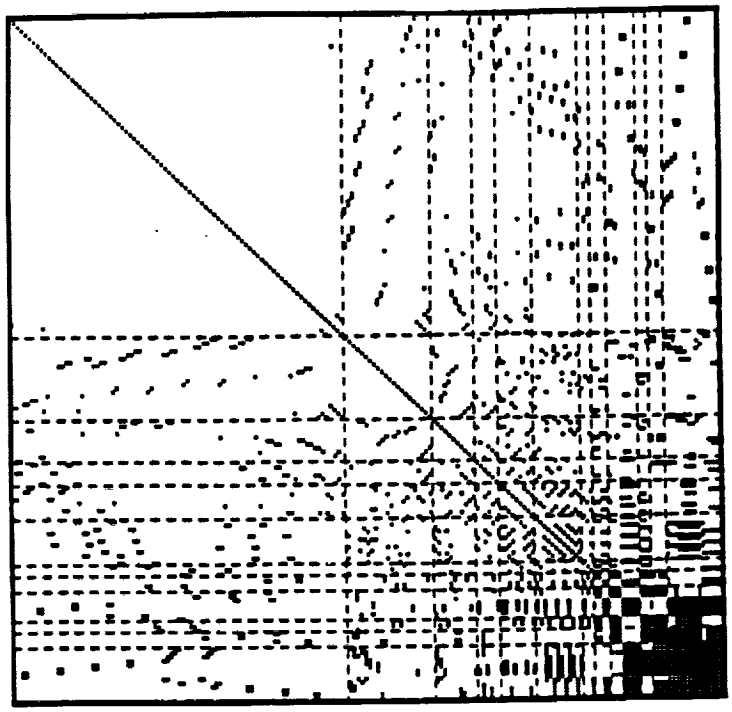


Figure 8: Five-point finite difference matrix for 10 by 10 grid. Matrix ordered by the minimum degree algorithm, then re-ordered and partitioned by RP1. Twelve factors result.

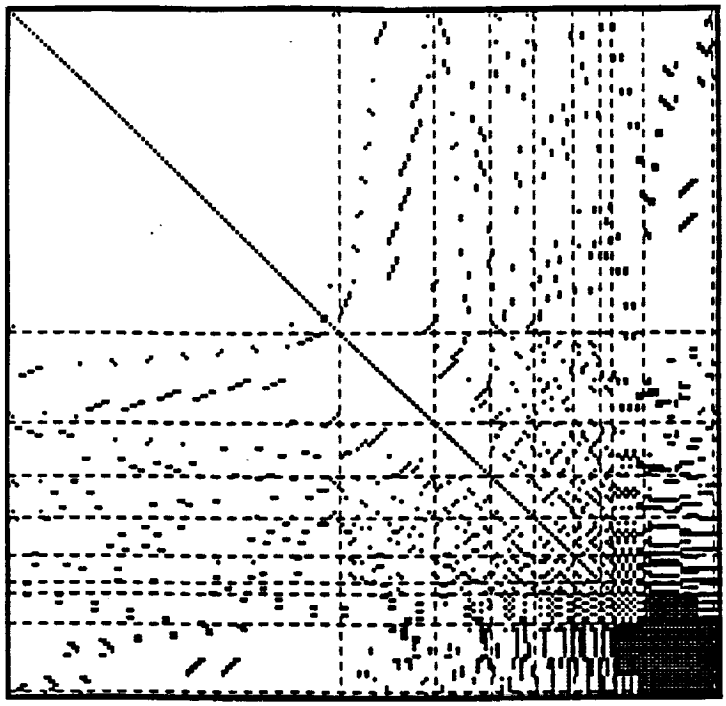


Figure 9: Five-point finite difference matrix for 10 by 10 grid. Matrix ordered by MLMD, then partitioned by P1. Nine factors result.

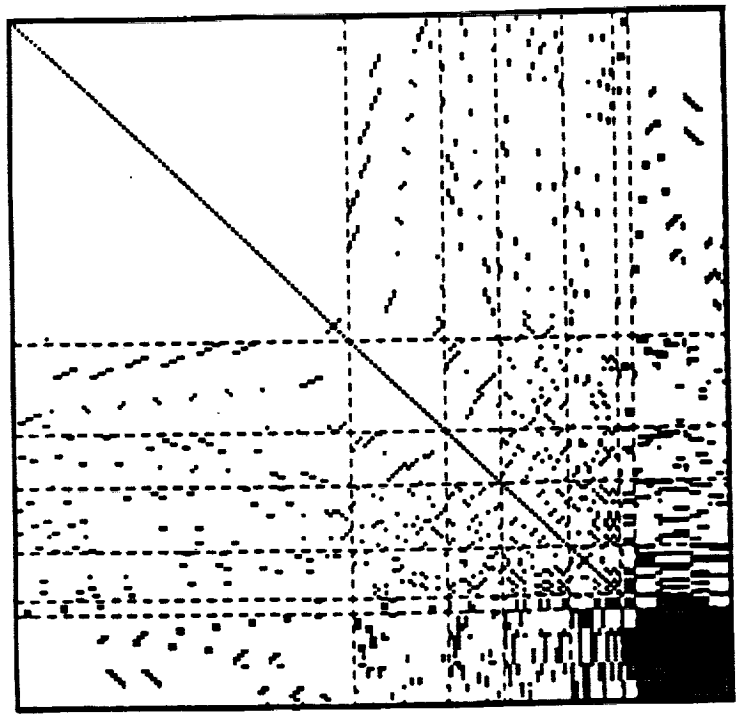


Figure 10: Five-point finite difference matrix for 10 by 10 grid. Matrix ordered by MLMD, then re-ordered and partitioned by RP1. Seven factors result.

same topology but where only seven factors are required, as illustrated in Figure 10.

Yet another way to look at the same information is to view the elimination trees associated with these matrices. Figure 11 illustrates the elimination tree associated with the 10 by 10 finite difference matrix ordered by MLMD and partitioned by P1. The corresponding matrix for this tree is shown in Figure 9. The height of the elimination tree is thirty-two. But there is a no-fill partition with only nine factors, which makes it possible to aggregate elements from many levels into a single partition. If the matrix is re-ordered once more using RP1 and then partitioned, the height of the tree and its shape remain identical, as illustrated in Figure 12, but greater grouping from among different levels is possible, resulting in only seven factors. In these figures, the elements from each partition are distinguished by using different shapes as well as shading.

To illustrate the very different kinds of tree shapes attainable, Figure 13 illustrates the partitioned factorization tree associated with the primary MLMD ordering of the size 118 power system example, re-ordered and partitioned by RP1.

The reader may wonder about the relationship of the present work and earlier work on exploitation of dense submatrices on sparse elimination. This work is all related to clique structures in the graph of L. (A clique is vertex subset that is completely interconnected. A maximal clique is a clique that is not a proper subset of another clique. A simplicial clique of $G(L+L^T)$ is a clique all of whose members have exactly the same nonclique neighbors.) Pothen and other authors give a number of applications of clique and clique tree representations of sparse matrix factors. [12] The partitioning of this paper is not the same as the partitioning into simplicial cliques or supernodes that has appeared previously. Indeed, all members of a simplicial clique of $G(L)$ are included in the same factor by Algorithms RP1 and RP2, but the converse is not true. Several simplicial cliques may belong to the same factor, as illustrated in Figure 13. The ten nodes denoted by shaded squares in this figure can all be included in a single factor, but they do not constitute a simplicial clique. Figure 14 illustrates the clique tree corresponding to the elimination tree of Figure 13.

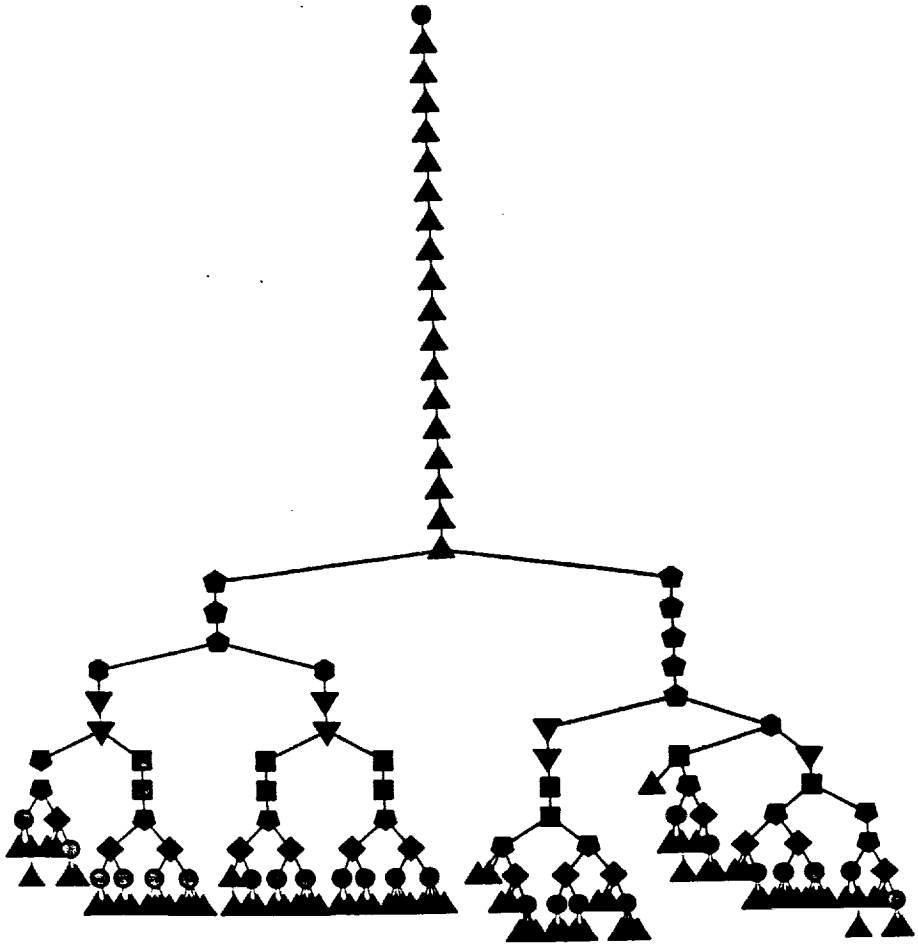


Figure 11: Elimination tree for finite difference grid matrix with MLMD primary ordering and partitioned using P1. The nine partitions are denoted by different shapes and shadings of the nodes (a tenth partition involving the last node only is trivial).

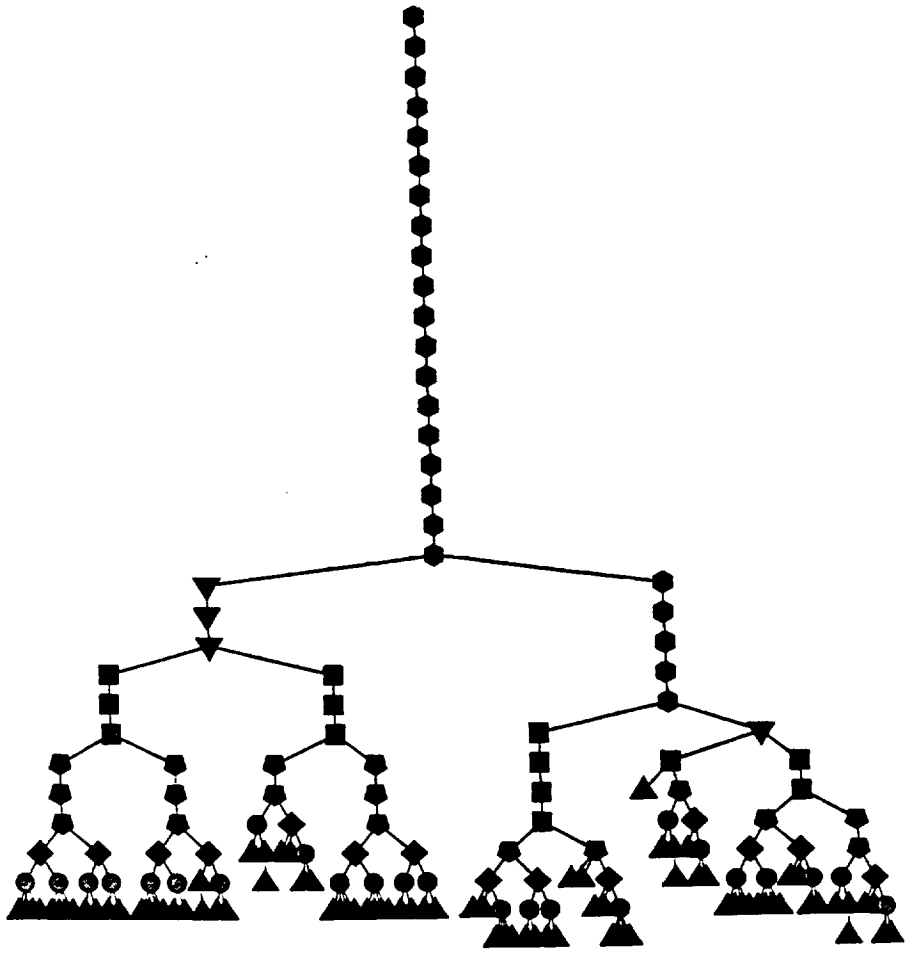


Figure 12: Elimination tree for finite difference grid matrix with MLMD primary ordering, re-ordered and partitioned using RP1. Seven factors.

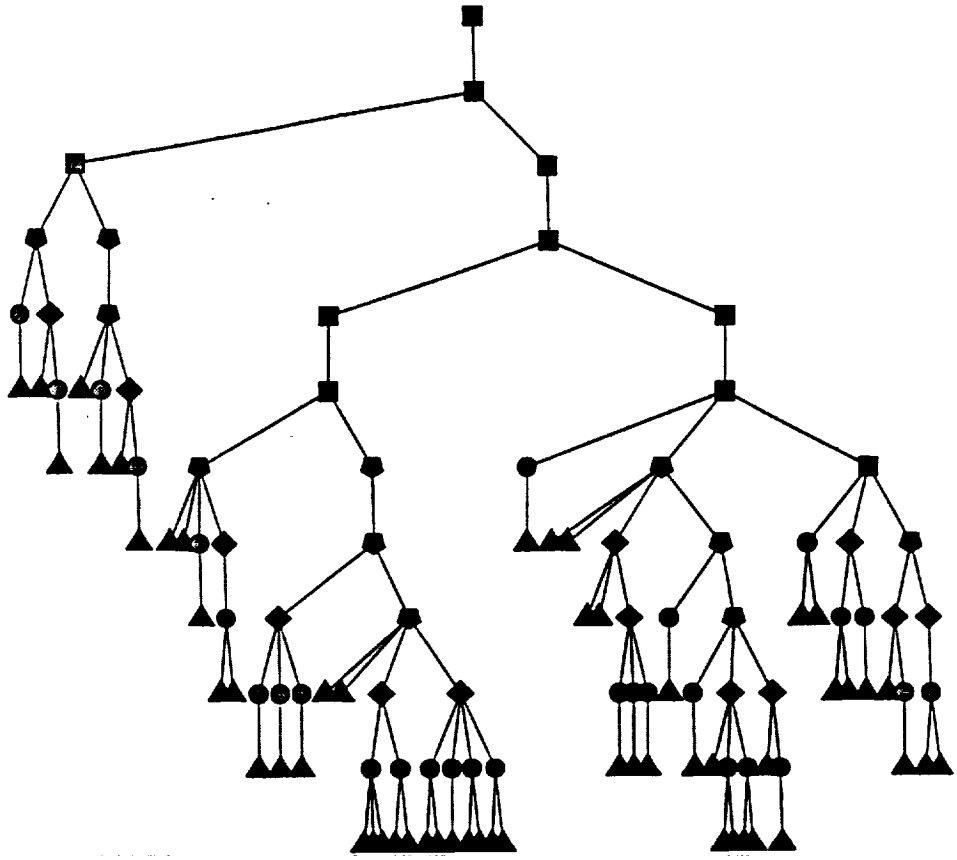


Figure 13: Elimination tree for 118 node power system matrix with MLMD primary ordering, re-ordered and partitioned using RP1. Five factors.

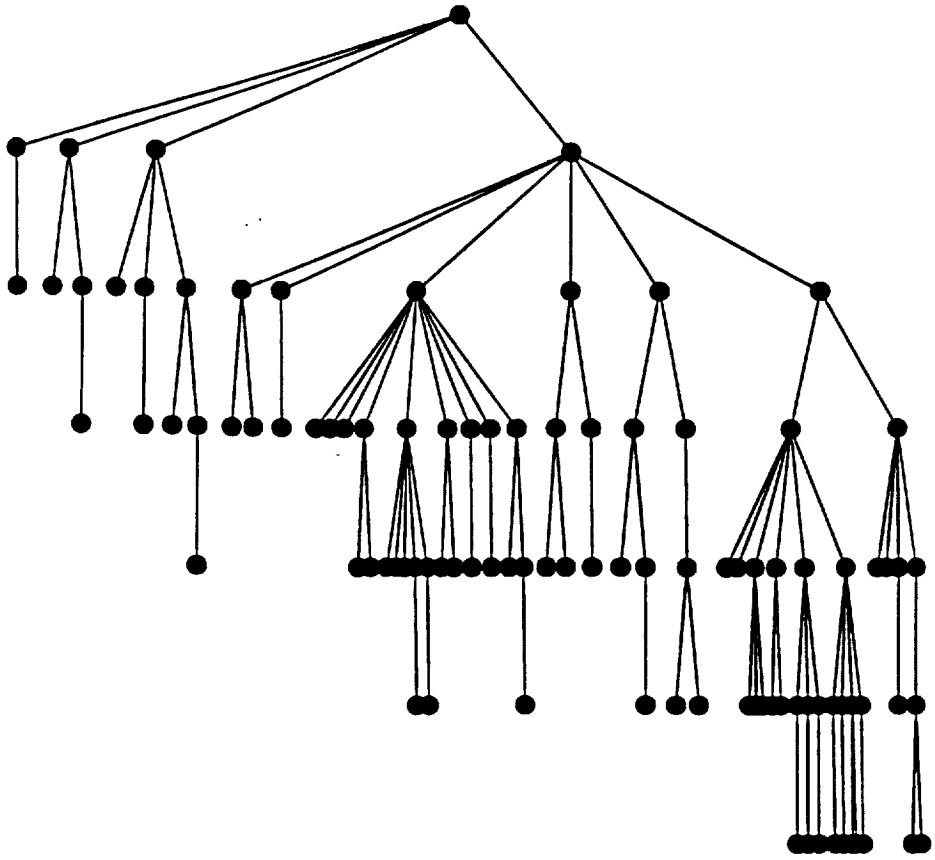


Figure 14: Clique tree for 118 node power system matrix with MLMD primary ordering, re-ordered and partitioned using RP1. This matrix has 99 maximal cliques and seven levels.

Table 1: Comparison of the number of partitions using five power system matrices, three initial ordering algorithms, and simple partitioning (P1) or partitioning after reordering with RP1.

	Min. Degree		MMD		MLMD	
Size	P1	RP1	P1	RP1	P1	RP1
118	53	14	10	10	6	5
352	132	21	13	12	8	8
707	213	26	23	18	11	10
1084	309	26	33	24	14	11
1993	563	35	41	25	15	15

Table 2: Comparison of the number of partitions using five-point finite difference matrices for a square grid, three initial ordering algorithms, and simple partitioning (P1) or partitioning after reordering with RP1.

	Min. Degree		MMD		MLMD	
Grid Size	P1	RP1	P1	RP1	P1	RP1
5 by 5	10	7	6	6	5	5
10 by 10	20	12	15	11	9	7
15 by 15	20	12	16	14	11	8

Conclusions

An algorithm for no-fill partitioning of lower triangular matrices with the fewest possible factors has been presented and proven to be optimal. This should reduce the number of steps required for the parallel solution of sparse triangular matrices when the same matrix is repeatedly used.

We have done a number of experiments with the triangular factors of sparse matrices, A. The structure of these factors is influenced by the ordering of rows and columns of A. (This ordering is chosen to reduce fills during the factorization process.) For such triangular factors, the ordering used in the factorization strongly influences the benefits attainable by the proposed algorithm. If L is constructed with the minimum degree algorithm, the minimum no-fill partition of L (without a change of ordering) tends to have a large number of factors. This number can be reduced significantly with the proposed re-ordering and partitioning algorithm. The same is true if the primary ordering is MMD. The results are better in this case than those obtained with primary minimum degree orderings. If A is ordered by the MLMD algorithm, the number of factors obtained without a change in the ordering is considerably smaller. The re-ordered partitioning algorithm can reduced further the number of factors, but its effect is not as dramatic as in the minimum degree case. Nevertheless, the total number of factors when the MLMD ordering of A is used is smaller than that if the minimum degree or MMD algorithm is used.

We also have provided a graphic illustration of the effect of partitioning on elimination trees and on clique trees. All the experiments and illustrations were done with the aid of the first author's Sparse Matrix Manipulation System [1].

References

[1] F. L. Alvarado. Manipulation and visualization of sparse matrices. *ORSA Journal on Computing* 2, (1990), pp. 180–207.

[2] F. L. Alvarado, D. Yu and R. Betancourt. Ordering schemes for partitioned sparse inverses. SIAM Symposium on Sparse Matrices, Salishan Lodge, Gleneden Beach, Oregon, May 22–24, 1989.

[3] F. L. Alvarado, D. Yu and R. Betancourt. Partitioned sparse A^{-1} methods. *IEEE Transactions on Power Systems* 5, (1990), pp. 452–459.

[4] E. Anderson. Parallel implementation of preconditioned cojugate gradient methods for solving sparse systems of linear equations. Technical Report 805, Center for Supercomputer Research and Development, Univ. of Illinois, 1988.

[5] C. Ashcraft and R. Grimes, The influence of relaxed supernode partition on the multifrontal method. *ACM Transactions on Mathematical Software* 15 (1989), pp. 291–309.

[6] R. Betancourt. An efficient heuristic ordering algorithm for partial matrix refactorization. *IEEE Transactions on Power Systems* 3 (1988), pp. 1181–1187.

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

[8] I. S. Duff and J. K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software* 9 (1983), pp. 302–325.

[9] M. K. Enns, W. F. Tinney and F. L. Alvarado. Sparse matrix inverse factors. *IEEE Transactions on Power Systems* 5 (1990) pp. 466–472.

[10] S. W. Hammond and R. Schreiber. Efficient ICCG on a shared memory multiprocessor. Technical Report 89.24, Research Institute for Advanced Computer Science, 1989.

[11] J.W. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software* 11 (1985), pp. 141–153.

[12] A. Pothen and C. Sun, Compact clique tree data structures in sparse matrix formulations. Computer Sciences Technical Report Number 897, December 1989, The University of Wisconsin, Madison.

[13] R. Schreiber. A new implementation of sparse gaussian elimination. *ACM Transactions on Mathematical Software* 9 (1983), pp. 302–325.

[14] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. IEEE* 55 (1967), pp. 1801–1809.