

ORNL/TM-11040

OAK RIDGE
NATIONAL
LABORATORY

MARTIN MARIETTA

A Fast Algorithm for Reordering Sparse Matrices for Parallel Factorization

John G. Lewis
Barry W. Peyton
Alex Pothier

OAK RIDGE NATIONAL LABORATORY

CENTRAL RESEARCH LIBRARY

DISSEMINATION SECTION

400N ROOM 112

LIBRARY LOAN COPY

DO NOT TRANSFER TO ANOTHER PERSON

If you wish someone else to see this
report, send in name with report and
the library will arrange a loan.

10/18/88 10:00 AM

OPERATED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

Printed in the United States of America. Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road, Springfield, Virginia 22161
NTIS price codes—Printed Copy: A04; Microfiche A01

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Engineering Physics and Mathematics Division

Mathematical Sciences Section

**A FAST ALGORITHM FOR REORDERING SPARSE MATRICES
FOR PARALLEL FACTORIZATION**

John G. Lewis †
Barry W. Peyton ††
Alex Pothen †††

† Scientific Computing and Analysis Division
Boeing Computer Services
P.O. Box 24346
Seattle, WA 98124

†† Mathematical Sciences Section
Oak Ridge National Laboratory
P.O. Box 2009, Bldg. 9207-A
Oak Ridge, TN 37831-8083

††† Department of Computer Science
The Pennsylvania State University
University Park, PA 16802

Date Published: January, 1989

Research was supported by the U.S. Air Force Office of Scientific Research under grants F49620-87-C-0037 and AFOSR-88-0161, by National Science Foundation grant CCR-8701723, by NSF equipment grant CCR-8705110, and by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy.

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
operated by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC-05-84OR21400



Contents

1	Introduction	1
2	The Jess and Kees Method	3
3	The Maximal Cliques of Filled Graphs	6
4	Clique Trees	12
5	Reduced Clique Trees	15
6	A Parallel Reordering Algorithm	24
7	Empirical Results	28
8	Concluding Observations	32
9	References	33

A FAST ALGORITHM FOR REORDERING SPARSE MATRICES FOR PARALLEL FACTORIZATION

John G. Lewis
Barry W. Peyton
Alex Pothen

Abstract

Jess and Kees (1982) introduced a method for ordering a sparse symmetric matrix A for efficient parallel factorization. The parallel ordering is computed in two steps. First, the matrix A is ordered by some fill-reducing ordering. Second, a parallel ordering of A is computed from the filled graph that results from factoring A using the initial fill-reducing ordering. Among all orderings whose fill lies in the filled graph, this parallel ordering achieves the minimum number of parallel steps in the factorization of A . Jess and Kees did not specify the implementation details of an algorithm for either step of this scheme. Liu and Mirzaian (1987) designed an algorithm implementing the second step, but it has time and space requirements higher than the cost of computing common fill-reducing orderings.

We present here a new fast algorithm that implements the parallel ordering step by exploiting the *clique tree* representation of a chordal graph. We succeed in reducing the cost of the parallel ordering step well below that of the fill-reducing step. Our algorithm has time and space complexity linear in the number of *compressed subscripts* for L , i.e., the sum of the sizes of the maximal cliques of the filled graph. Empirically we demonstrate running times nearly identical to Liu's heuristic Composite Rotations algorithm that approximates the minimum number of parallel steps.

1. Introduction

An important problem in the design of parallel algorithms for the direct solution of sparse, symmetric, positive definite systems is reordering the given matrix A so that the reordered matrix can be factored efficiently in parallel. Jess and Kees [9] suggested a modular approach to this problem. First, a permutation matrix P is computed such that the factor L is sparse, where $PAP^T = LL^T$; the adjacency graph of the resulting filled matrix $F = L + L^T$ is a chordal graph. Second, a parallel ordering is computed from the chordal filled graph by successively removing maximum independent sets of simplicial (non-deficient) nodes from the graph. If F were viewed as a matrix to be factored, then the computed ordering permits the parallel factorization of F in the minimum number of steps, under the constraint that no additional fill in F is permissible. Alternatively, the computed ordering achieves the minimum number of parallel steps in factoring A for all orderings whose fill lies in the filled graph resulting from the original sequential ordering.

Liu [13] showed that the Jess and Kees method finds a parallel ordering that has the shortest elimination tree over all perfect elimination orders of the filled graph. The elimination tree captures the dependencies among the columns of A , and its height is an appropriate measure of parallel complexity in a computing environment in which the elimination of a column can be assumed to be a constant-time operation. In addition, the elimination tree height is closely related to other measures of parallel complexity (Liu [12]), and provides a simple but effective criterion for parallelism in several models of parallel factorization.

This modular approach only provides the minimum number of parallel steps for parallel orderings with fill constrained by the initial fill-reducing ordering. Some justification for this restricted approach is provided in [18], where it is shown that the problem of computing an ordering of A with the shortest elimination tree is NP-complete, and hence intractable.

In the Jess and Kees approach, the parallel reordering algorithm is applied as a post-processing to an initial fill-reducing ordering. It is important that this post-processing not markedly increase the complexity of the overall reordering scheme. Jess and Kees did not specify the implementation details of an algorithm for the second step of their scheme. Liu and Mirzaian [15] have designed an algorithm that implements this step, but its complexity is unsatisfactory. Its time and space requirements are linear in the size of the filled matrix $F = L + L^T$, and the post-processing is often more expensive than the rest of the ordering process. Indeed, more storage may be needed for the post-processing than for the factorization of the matrix. Recognizing these requirements as unacceptably large, Liu [13] designed a heuristic Composite Rotations

algorithm that attains *nearly* minimal tree height. This algorithm is based on the concept of elimination tree rotations and has space and time requirements that are almost linear in the size of the original matrix A . In [13] Liu empirically compares the Liu and Mirzaian implementation of the Jess and Kees method with the tree rotations heuristic and demonstrates that the heuristic is an excellent approximation in practice and is also much faster. Indeed, this heuristic also minimizes tree height on all but two of his test problems.

The motivation for this work is to reduce the complexity of computing the Jess and Kees reordering. By representing the chordal graph as a *clique tree* we reduce both the time and storage required for the reordering algorithm to terms linear in the sum of the sizes of the maximal cliques of the filled graph. In terms of sparse matrix data structures, our algorithm has complexity bounded by the size of the array of subscripts for L after mass elimination compression. Empirically, the cost of our Jess and Kees implementation is essentially the same as the cost of Liu's heuristic. This cost is small enough that finding the equivalent ordering with a shortest tree can be viewed as merely another small part of the reordering process.

The organization of our paper is as follows. We begin in §2 by describing the Jess and Kees method and show how the *maximal cliques* of the filled graph can be used to design an efficient implementation. This will provide the reader with an overview of our algorithm. The next sections present the details of an efficient practical implementation that is obtained from the compressed row subscript lists of the sparse Cholesky factor. The first tool we need is a description of the maximal cliques of the filled graph; an algorithm for identifying the maximal cliques from the subscript lists of the Cholesky factor is given in §3. In §4 we extend the maximal clique algorithm to arrange the maximal cliques in a *clique tree* and present some important properties of the clique tree. The following section §5 discusses how the maximal clique structure and the clique tree change as nodes are eliminated and, in particular, how clique trees simplify determining these changes. At this point all necessary tools are in place to design an efficient parallel reordering algorithm, which is presented in §6. A complexity analysis shows that our algorithm is linear in the size of the compressed version of the row subscripts. In §7, we present experimental results on our reordering algorithm and empirical comparisons with the Liu and Mirzaian algorithm and Liu's heuristic algorithm. The last section §8 contains some concluding observations on clique trees and directions for further work.

We denote the adjacency graph of F , the *filled graph*, by G_F . The number of nodes in G_F is the order n of the matrix A . The size of the adjacency lists of G_F (twice the number of edges in G_F , also twice the number of offdiagonal nonzero entries in L), we denote by $\eta(F)$. The number of maximal cliques in G_F will be denoted by m . Let

$\{K_1, K_2, \dots, K_m\}$ be the set containing the maximal cliques of G_F . The total size of the maximal cliques of the graph will be denoted by q and is defined by $q := \sum_{i=1}^m |K_i|$.

2. The Jess and Kees Method

In this section we describe the Jess and Kees method for computing a good ordering for parallel sparse factorization and provide an overview of our clique tree algorithm that implements this method.

We begin with a matrix PAP^T , where P is a permutation matrix chosen to preserve sparsity, and we assume that a symbolic factorization has been computed. Note that the ordering $\{1, 2, \dots, n\}$ is a perfect elimination (no-fill) ordering for the *filled matrix* $F = L + L^T$. We want to find a reordering that is more suitable for the parallel factorization of A . If we find a good parallel ordering Q from among the perfect elimination orderings of F , then we incur no additional fill (compared to PAP^T) when A is ordered by the composition of the two orderings, QP .

We now seek to motivate the Jess and Kees method. In the elimination graph model, a node v in G_F can be eliminated without causing any fill only if its adjacency set $adj(v)$ is a clique; such a node is called *simplicial* or *non-deficient*. To obtain a perfect elimination ordering of F appropriate for parallel factorization, we could attempt to eliminate all the simplicial nodes of G_F in one step. This is not feasible, however, because of the dependencies between the columns that these nodes represent.

To see this, consider the submatrix Cholesky method for computing the numeric factorization. The elimination of a node numbered j in the graph is equivalent to computing the j -th column of L from the j -th column of A by scaling each element by the square root of its diagonal element, and then subtracting a multiple of this column of L from all higher numbered columns k of A with $l_{kj} \neq 0$. Thus, column k of L cannot be computed before column j of L is computed. This implies that in parallel factorization, columns j and k of L , $j < k$, may be computed in the same step only if $l_{kj} = 0$.

In the graph model of the factorization, two nodes j and k can be eliminated at the same step only if no edge joins them, or equivalently, only if the nodes j and k are *independent*. Therefore a perfect elimination ordering of G_F suitable for parallel factorization generally cannot eliminate all of its simplicial nodes in a single step, but only an independent subset of the simplicial nodes. The Jess and Kees method is a typical greedy algorithm: it eliminates a *maximum independent set* of simplicial nodes at each step, as described below.

repeat
 choose a maximum independent set of simplicial nodes
 number these nodes consecutively and eliminate them from the graph
until
 all nodes are eliminated.

Consider an example: If F is a tridiagonal matrix, then G_F corresponds to a chain graph, and the only simplicial nodes are the two nodes at either end of the graph. The Jess and Kees method would eliminate both of these nodes in a single step, which results in a smaller chain graph. This results in the ‘burn at both ends’ algorithm, which has the smallest number of parallel steps for factoring a tridiagonal matrix with no fill.

A fast implementation of the Jess and Kees method requires an efficient way to detect simplicial nodes and to identify a maximum independent set of simplicial nodes. As noted earlier, Jess and Kees did not specify how these operations should be performed. In the Liu and Mirzaian implementation [15], the filled graph is represented explicitly by the adjacency lists of the nodes, and the nodes are assumed to be in a perfect elimination order. Then a simplicial node v is identified by a test involving the degrees and monotone degrees of nodes. (The monotone degree of a node is the size of the set of nodes adjacent to v and numbered higher than v , and is denoted by $|Madj(v)|$. These degree measures must be maintained throughout the elimination process, and this requires updating these quantities for all uneliminated neighbors of each node when it is eliminated. Thus the cost of detecting simplicial nodes in their implementation is $\mathcal{O}(n + \eta(F))$. Some of this work can be eliminated by considering the degree measures as functions of cliques of simplicial nodes [17]; however, the worst-case complexity does not change. A better approach is obtained by using the maximal clique structure of the chordal graph directly to identify simplicial nodes.

We exploit the maximal clique structure of the chordal filled graph and its clique tree representation to obtain a more efficient implementation of the Jess and Kees method. Recall that a clique is a set of nodes that are all pairwise adjacent. A *maximal clique* is a clique to which no other node in the graph can be added while preserving the pairwise adjacency property. Our algorithm is based on the following key properties of the maximal clique structure of chordal graphs, each of which will be verified later:

- a node is simplicial if and only if it is contained in exactly one maximal clique [5,20].
- a simplicial node is independent of the simplicial nodes in other maximal cliques, but not independent of simplicial nodes in the same maximal clique.
- the maximal cliques can be organized into a clique tree structure [1,2,6,16,20,21].
- the elimination of simplicial nodes changes the maximal clique structure of the reduced graph; the changes cause simple local changes in the clique tree of the graph.

The first of these properties is the basis for detecting simplicial nodes. It is clear from the second property that a *maximum* independent set of simplicial nodes can be found by choosing one simplicial node from each maximal clique that contains any simplicial nodes. Thus, both independence and simpliciality of nodes can be characterized from the maximal clique structure. The use of clique trees will enable us to track the clique structure as it evolves during the elimination process. In addition, the clique tree representation of the filled graph is much more compact than its adjacency list representation; indeed, sometimes it is even more compact than the adjacency list representation of the original graph.

Our fast implementation of the Jess and Kees method is outlined below.

repeat

 choose one simplicial node from each maximal clique with at least one
 eliminate this maximum independent set of simplicial nodes
 update the clique tree

until

 all nodes are eliminated.

By recognizing and maintaining the maximal clique structure, we reduce the tests for simpliciality and independence from operations that occur on the nodes and edges of the filled graph to operations that occur on the maximal cliques. The fundamental tools we use are first, maximal cliques, as discussed in §3; second, the clique tree, which is described in §4; and third, the evolution of the clique tree as it changes during elimination, considered in §5. Finally, in §6, we use these results to state our parallel reordering algorithm in detail.

We assume the following context in the next three sections. A sparse symmetric matrix has been ordered by a fill-reducing procedure such as the minimum degree

algorithm. To simplify notation, we assume that the matrix A has been renumbered in the fill-reducing ordering. Under this assumption, the natural ordering is a perfect elimination ordering for the filled matrix F . In addition, we assume that an efficient symbolic factorization procedure has been used to compute the row subscripts of the nonzeros in the columns of the Cholesky factor L , and that the row subscripts for each column are stored in ascending order. For simplicity, we assume that the adjacency graph of A is connected; otherwise, the generalizations from clique trees to clique forests are straightforward.

3. The Maximal Cliques of Filled Graphs

In this section we establish some fundamental properties of the maximal cliques of chordal graphs. We then present an algorithm for identifying and representing the maximal cliques using the row subscript lists obtained from a symbolic factorization. We begin with the characterization of a simplicial node in terms of maximal cliques, which exhibits the fundamental connection between the two concepts. It is worth noting that Proposition 1 holds for any graph.

Proposition 1. *A node is simplicial if and only if it is contained in only one maximal clique.*

Proof: If a node v is simplicial, then, by definition, $K = adj(v) \cup \{v\}$ is a clique. K is a maximal clique because any $z \notin K$ is not adjacent to v and so $\{z\} \cup K$ is not a clique. K is the only maximal clique containing v since any other maximal clique containing v must contain some $z \notin K$. Since z and v are not adjacent, they cannot be in the same clique.

Conversely, suppose v is not simplicial. We show that v must lie in at least two maximal cliques. By definition, $adj(v)$ is not a clique, so there exist two nodes u and $w \in adj(v)$ that are not mutually adjacent. Since $\{u, v\}$ is a clique, we can choose some maximal clique K_u containing $\{u, v\}$. Similarly there exists some maximal clique K_w containing $\{v, w\}$. Since u and w are not adjacent, $u \notin K_w$ and $w \notin K_u$, and so K_u and K_w are distinct maximal cliques, both containing v . ■

Duff and Reid [5] have employed this characterization of a simplicial node as a theoretical tool. In [20] Tarjan and Yannakakis use the result in a scheme for partially reducing acyclic hypergraphs. To make this characterization a practical tool, we must have an efficient way to find and represent the maximal cliques. We proceed to show that the maximal cliques are already represented in the subscript lists produced by the symbolic factorization.

Proposition 2. *If K is a maximal clique in G_F and v is the lowest numbered node in K , then $K = \{v\} \cup \text{Madj}(v)$.*

Proof: By definition, the monotone adjacency set $\text{Madj}(v)$ consists of all nodes adjacent to v and numbered higher than v . $K' = \{v\} \cup \text{Madj}(v)$ is a clique because the numbering is a perfect elimination ordering for G_F . Consider some node z in $K - K'$. By the choice of v , z must be numbered higher than v . But z cannot then be adjacent to v , else it belongs to K' . This contradiction shows that $K \subseteq K'$. Equality follows because K is maximal. ■

Proposition 2 implies that the maximal cliques of G_F can be represented as $K(v_1)$, $K(v_2)$, \dots , $K(v_m)$, where $K(v)$ specifies the maximal clique $\{v\} \cup \text{Madj}(v)$ generated by the lowest numbered or *representative* node v . If a node is not the representative of any maximal clique, we call it a *non-representative*. Since the set $\text{Madj}(v)$ is precisely the row subscripts of the nonzero entries in column v of L , the maximal cliques of G_F are given implicitly by the subscript lists. Each maximal clique consists of its representative together with the subscripts in the column corresponding to the representative.

In Figure 1, a perfect elimination ordering of a chordal graph is shown. The reader can verify that its maximal cliques are $K(1) = \{1, 2, 3\}$, $K(3) = \{3, 4, 6, 7\}$, $K(5) = \{5, 6\}$, and $K(6) = \{6, 7, 8\}$.

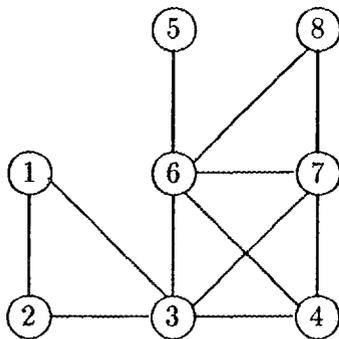


Figure 1: A perfect elimination ordering of a chordal graph.

We use this characterization to identify the maximal cliques from the row subscript lists generated by symbolic factorization. The problem is to recognize which nodes are representatives; the solution is to recognize the nodes that are not representatives.

Proposition 3. *A node v is not a representative if and only if there exists a representative node z numbered lower than v , with $\{v\} \cup \text{Madj}(v) \subset \{z\} \cup \text{Madj}(z)$.*

Proof: If such a node z exists, then the clique $\{z\} \cup \text{Madj}(z)$ is a proper superset of $\{v\} \cup \text{Madj}(v)$ and hence $\{v\} \cup \text{Madj}(v)$ is not maximal.

Conversely, suppose there is no node z numbered lower than v with $\{v\} \cup \text{Madj}(v) \subset \{z\} \cup \text{Madj}(z)$. By Proposition 2, any maximal clique has the form $\{w\} \cup \text{Madj}(w)$ where w is its representative. Thus any maximal clique containing $\{v\} \cup \text{Madj}(v)$ contains only v and nodes numbered higher than v . But then v is the lowest numbered node in such a maximal clique and by Proposition 2, v is the representative that generates the maximal clique $K(v) = \{v\} \cup \text{Madj}(v)$. ■

The following Proposition provides an efficient test of whether $\{v\} \cup \text{Madj}(v) \subset \{z\} \cup \text{Madj}(z)$, using only the sizes of the subscript lists. The relevant quantities can be easily obtained when we recall that $\text{Madj}(z)$ is the subscript list for column z , and the monotone degree $Mdeg(z)$ is the size of this list.

Proposition 4. *Let $K(v)$ be a maximal clique, with representative v and nodes $v < w_1 < w_2 < \dots < w_p$. Then w_i is not a representative node for any maximal clique if $Mdeg(w_i) = p - i$.*

Proof: Since the nodes are in a perfect elimination order,

$$\begin{aligned} \text{Madj}(v) &\subseteq w_1 \cup \text{Madj}(w_1) \\ &\subseteq \{w_1, w_2\} \cup \text{Madj}(w_2) \\ &\dots \\ &\subseteq \{w_1, \dots, w_p\} \cup \text{Madj}(w_p). \end{aligned}$$

This implies that $Mdeg(v) \leq Mdeg(w_1) + 1 \leq Mdeg(w_2) + 2 \leq \dots \leq Mdeg(w_p) + p$. Assume $Mdeg(v) - i = p - i = Mdeg(w_i)$. It follows that equality holds for every set inclusion in the chain from $\{w_1, \dots, w_i\} \cup \text{Madj}(w_i)$ down to $\text{Madj}(v)$, so that these two sets are equal. Then by Proposition 3, w_i is not a representative node. ■

Although Proposition 4 is stated as a test on a single node, it should be clear from the proof that when w_i is non-representative, w_1, w_2, \dots, w_{i-1} are also non-representative. On the other hand, suppose that for some j , $i < j \leq p$, we have $Mdeg(w_j) > p - j$. Then $Mdeg(w_k) > p - k$ for each k , $j \leq k \leq p$. If we take j as the index of the lowest numbered node that fails the degree test, then the maximal clique

$K(v)$ has the structure

$$K(v) = \{v\} \cup \{w_1, \dots, w_{j-1}\} \cup \{w_j, \dots, w_p\},$$

where w_1, \dots, w_{j-1} are not representative and w_j, \dots, w_p cannot be judged as non-representative by examining $K(v)$. We use this result to reduce the work needed to apply Propositions 3 and 4 in an algorithm to determine the maximal cliques of G_F .

Our algorithm for determining the maximal cliques of G_F does so by finding which nodes are representatives and which nodes are not. Initially we mark all nodes as possible representatives, and process nodes in the given perfect elimination order. If a node v has not been marked as a non-representative by the time it is processed, no earlier node can satisfy the role of z in Proposition 3 and so we accept v as a representative. We then examine $Madj(v)$ and mark nodes satisfying the degree test in Proposition 4 as non-representatives, stopping when a node fails the degree test. An informal algorithm follows.

```
mark all nodes as possible representatives
for  $v = 1$  to  $n$  do
  if  $v$  is still a possible representative then
    /* $v$  is a representative*/
    mark the nodes in  $Madj(v)$  in order as non-representative
    until a node fails the degree test.
  endif
endfor
```

The marking procedure in the algorithm can be modified to make it faster, since it is redundant to mark a given node repeatedly as non-representative. We can terminate the marking procedure as soon as a previously marked node is encountered. That this is sufficient grounds for terminating the search is shown by the following proposition.

Proposition 5. *Let the non-representative nodes in $K(v)$ be w_1, \dots, w_{j-1} . Suppose that for some $i \leq j-1$, w_i is also non-representative in a maximal clique $K(u)$, where $u < v$. Then w_{i+1}, \dots, w_{j-1} are also non-representative nodes in $K(u)$.*

Proof: Let $K(u) = \{u\} \cup \{z_1, \dots, z_{l-1}\} \cup \{z_l, \dots, z_t\}$, with $w_i = z_k \in \{z_1, \dots, z_{l-1}\}$ appearing as a non-representative node in $K(u)$. Then, as we saw in the proof of

Proposition 4 and the discussion following it,

$$\begin{aligned} Madj(u) &= Madj(w_i) \cup \{z_1, \dots, z_{k-1}, w_i\} \\ Madj(w_i) &= \{w_{i+1}, \dots, w_{j-1}\} \cup Madj(w_{j-1}) \end{aligned}$$

that implies that

$$Madj(u) = \{z_1, \dots, z_{k-1}\} \cup \{w_i, w_{i+1}, \dots, w_{j-1}\} \cup Madj(w_{j-1}).$$

Hence, w_{j-1} is a non-representative node in $K(u)$. The same argument obviously holds for $\{w_{i+1}, \dots, w_{j-2}\}$ as well. ■

Proposition 5 implies that we can stop the search for non-representative nodes as soon as we find a node w_l which fails the degree test or has already been marked as non-representative. The modified algorithm for finding the representative nodes imposes the following structure on the maximal clique $K(v)$, where w_l is the last node examined in $K(v)$:

$$\begin{aligned} new(K(v)) &= \{v, w_1, \dots, w_{l-1}\} \\ anc(K(v)) &= \{w_l, \dots, w_p\} \\ first_anc(K(v)) &= w_l. \end{aligned}$$

The notation adopted for these sets will become more meaningful when we use these sets to organize the maximal cliques into a clique tree. The set $new(K(v))$ consists of the representative v together with the nodes marked as non-representative while examining $K(v)$. The set $anc(K(v))$ (read ‘ancestor set of $K(v)$ ’) consists of the nodes in $K(v)$ that either fail the degree test or were marked as non-representative in some other maximal clique $K(u)$, with $u < v$. The node $first_anc(K(v))$ (read ‘first ancestor node in $K(v)$ ’), the last node examined at $K(v)$, is also the lowest numbered node in $anc(K(v))$. It will play a crucial role in the clique tree structure.

Again it might be helpful to the reader to consider the new , anc partition for the maximal cliques in Figure 1. These sets are as follows:

$$\begin{aligned} new(K(1)) &= \{1, 2\} & anc(K(1)) &= \{3\} \\ new(K(3)) &= \{3, 4\} & anc(K(3)) &= \{6, 7\} \\ new(K(5)) &= \{5\} & anc(K(5)) &= \{6\} \\ new(K(6)) &= \{6, 7, 8\} & anc(K(6)) &= \emptyset. \end{aligned}$$

Algorithm 3.1.

```
clique_num := 0
for v := 1 to n do
    new_in_clique(v) := nil
    clique_count(v) := 0
endfor

for v := 1 to n do
    if new_in_clique(v) = nil then
        /*node v is a representative*/
        clique_num := clique_num + 1
        clique_size := |L*v|
        new_in_clique(v) := clique_num
        clique_count(v) := clique_count(v) + 1
        #_new(clique_num) := 1
        first_anc(clique_num) := nil

        for w ∈ L*v in increasing order do
            clique_count(w) := clique_count(w) + 1
            if first_anc(clique_num) = nil then
                clique_size := clique_size - 1
                if |L*w| = clique_size and new_in_clique(w) = nil then
                    #_new(clique_num) := #_new(clique_num) + 1
                    new_in_clique(w) := clique_num
                else
                    first_anc(clique_num) := w
                endif
            endif
        endfor
    endif
endfor
```

The partition of a maximal clique into the sets $new(K)$ and $anc(K)$ has two important properties that we note now for future use. First, the first ancestor node of K splits K into the ordered subsets, $new(K)$ and $anc(K)$; all nodes in $new(K)$ are numbered lower than the first ancestor, and the other nodes in $anc(K)$ are numbered higher than

the first ancestor. Second, the sets $\{new(K(v_1)), new(K(v_2)), \dots, new(K(v_m))\}$ form a partition of the nodes of the chordal graph, since every node is chosen as a representative or marked as a non-representative exactly once. We will use these properties later in defining a clique tree from the maximal cliques.

Algorithm 3.1 finds the representative nodes and the first ancestor nodes for each maximal clique in G_F . The maximal cliques are numbered from 1 to m , using the variable *clique_num*. The first ancestor node partitions each maximal clique K into the sets $new(K)$ and $anc(K)$. The procedure also computes *clique_count*(v), the number of maximal cliques containing v , and $\#_{new}(K)$, the number of nodes in $new(K)$. The former count will enable us to identify simplicial nodes, using Proposition 1. To emphasize the connection between the maximal cliques and the subscripts of the factor L , we use the notation L_{*v} to denote the subscripts of the nonzero entries in the v -th column of L , i.e., $Madj(v)$. The mapping $new_in_clique(v) = K$ means that $v \in new(K)$. Later, we use this mapping together with the first ancestor nodes to compute the clique tree.

The complexity of the algorithm is easily analyzed. The initializations require $\mathcal{O}(n)$ time. The processing of the nodes, excluding the examination of the $Madj(v)$ sets, also cost $\mathcal{O}(n)$ time. Since the $Madj(v)$ sets are examined only for representative nodes, and the cost of the operations associated with a node $w \in Madj(v)$ is a constant, the total cost of the latter operations is proportional to the size of the maximal cliques, which is $\mathcal{O}(q)$. Thus the overall complexity is $\mathcal{O}(n + q)$.

4. Clique Trees

In this section we exhibit how the maximal cliques of a chordal graph can be joined together in a meaningful tree structure. The concept of a *clique tree* has previously appeared in [1,16,20], and proofs that a clique tree can be constructed if and only if the graph is chordal can be found in the first two references. Our goals here are, first, to demonstrate that a clique tree can be constructed from Algorithm 3.1, and, second, to elucidate some properties of clique trees. We will use these properties in the following section to show how clique trees express the changing clique structure as simplicial nodes are eliminated from the filled graph by the Jess and Kees algorithm.

Recall that the sets $\{new(K_1), new(K_2), \dots, new(K_m)\}$, defined with respect to Algorithm 3.1, partition the nodes of the graph because every node is either chosen as a representative or marked as non-representative exactly once. As a result, the following *parent* function is well-defined if $anc(K) \neq \emptyset$:

$$parent(K) = \text{the clique } P(K) \text{ such that } first_anc(K) \in new(P(K)).$$

The *clique tree* has a node corresponding to each maximal clique, and an edge joining each maximal clique to its parent. The root of the clique tree has $anc(K) = \emptyset$. We let $P(K)$ denote the parent of a maximal clique K . Let $first_anc(K) = w$. Then $w \in new(P(K))$ implies that $w < first_anc(P(K))$. Hence there cannot be any cycles of the form $K, P(K), P(P(K)), \dots, K$, which shows that the *parent* function does indeed define a tree.

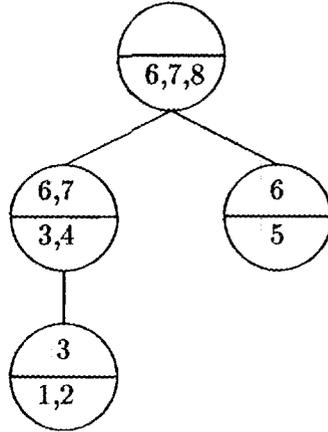


Figure 2: The clique tree of the chordal graph with perfect elimination order shown in Figure 1.

The clique tree corresponding to the perfect elimination order of the chordal graph in Figure 1 is shown in Figure 2. The partition of each maximal clique into the *new* and *anc* sets is shown by writing the nodes in $anc(K)$ above the nodes in $new(K)$. The first ancestor node of each maximal clique is the first node listed in $anc(K)$, and the reader can verify that the first ancestor node of each maximal clique is a *new* node in its parent clique.

We now prove some elementary relationships between a child clique and its parent.

Proposition 6. *Let w be the first ancestor node in a maximal clique K and let $P(K)$ be the parent of K . Then*

- $w \in new(P(K))$
- $anc(K) \subset P(K)$,
- $new(K) \cap P(K) = \emptyset$.

Proof: The first statement follows immediately from the definition of a parent clique. To prove the second, note that by definition of the first ancestor node we have $anc(K) \subset$

$\{w\} \cup \text{Madj}(w)$. Because $w \in \text{new}(P(K))$, either w is the representative of $P(K)$ or w was marked as a non-representative in $P(K)$ by Algorithm 3.1. In either case $\{w\} \cup \text{Madj}(w) \subseteq P(K)$. Thus $\text{anc}(K) \subset P(K)$.

To prove the third statement, suppose there exists a node $y \in \text{new}(K) \cap P(K)$. Since $y \in \text{new}(K)$, and w is the first ancestor of K , $y < w$ by the definition of the sets $\text{new}(K)$ and $\text{anc}(K)$. Now $w \in \text{new}(P(K))$ and $y < w$ imply that $y \in \text{new}(P(K))$, again by the definition of the sets $\text{new}(P(K))$, and $\text{anc}(P(K))$. Then $y \in \text{new}(P(K)) \cap \text{new}(K)$, and this contradicts the fact that the *new* sets partition the nodes. ■

Algorithm 3.1 partitions each maximal clique into the sets $\text{anc}(K)$ and $\text{new}(K)$. The previous result shows that the clique tree also splits each maximal clique K into the same two sets: The nodes in $\text{new}(K)$ are not in the parent; each node in $\text{anc}(K)$ is also in $P(K)$.

Proposition 6 gives a local characterization of the *new*, *anc* partition. The following theorem presents a global interpretation of this partition—viewed from the root R of the tree, the *new* nodes appear first in the clique which marked them as *new*, and the *anc* nodes appear first in some ancestor of this clique.

Theorem 1. *Given a maximal clique K and the partitioning $K = \text{new}(K) \cup \text{anc}(K)$,*

- *a node $u \in \text{new}(K)$ belongs to another maximal clique D only if D is a descendant of K in the clique tree;*
- *a node $u \in \text{anc}(K)$ belongs to $\text{new}(A)$, where A is some ancestor of K in the clique tree, and u also belongs to all maximal cliques on the path between K and A .*

Proof: For the first part, assume that there is some maximal clique $J \neq K$ which is not a descendant of K and yet contains a node u in $\text{new}(K)$. Choose J to be a maximal clique at the least distance from the root R satisfying this property. The node u cannot be a member of $\text{new}(J)$, lest we contradict the partitioning principle. However, $u \notin \text{new}(J)$ implies $u \in \text{anc}(J)$, which, by Proposition 6, implies that u is in J 's parent, $P(J)$. But $P(J)$ is then closer to the root than J , contradicting the choice of J . (The special case where $J = R$ leads to a contradiction because $\text{anc}(J) = \emptyset$.)

To prove the second part, let $u \in \text{anc}(K)$. Since the *new* sets partition the nodes, $u \in \text{new}(A)$ for some maximal clique A . It follows from the first part of the theorem that K must be a clique in the subtree rooted at A . Thus A is an ancestor of K . It remains only to prove that u belongs to all the intermediate ancestors $P(K) = I_1, P(P(K)) = I_2, \dots, I_p = A$. Suppose not. By Proposition 6, $u \in P(K) = I_1$. Hence there must exist a least index $2 \leq j < p$ such that $u \in I_{j-1}$ and yet $u \notin I_j$. By the

same Proposition, $u \in I_{j-1}$ implies either $u \in \text{new}(I_{j-1})$ or $u \in \hat{P}(I_{j-1}) = I_j$. The assumption that $u \notin I_j$ then requires u to belong to $\text{new}(I_{j-1})$, which again violates the fact that the *new* sets form a partition. ■

Recall that if $\text{first_anc}(K) = w$, then $w \in \text{new}(P(K))$, and hence w does not belong to any other ancestor of K . Thus, by our construction of a clique tree, the parent $P(K)$ is the only ancestor clique that contains the set $\text{anc}(K)$; all other ancestors of K contain only a proper subset of $\text{anc}(K)$. This property will be extremely useful in efficiently updating the clique tree in the Jess and Kees method.

5. Reduced Clique Trees

In the preceding sections we have presented the maximal clique structure of a filled graph and its clique tree representation. The Jess and Kees method produces another perfect elimination ordering of the filled graph in a sequence of steps. At each step a maximum independent set of simplicial nodes (and the edges incident on them) are removed from the graph, producing a sequence of reduced (elimination) graphs. Since we delete only simplicial nodes, no new edges are created in the elimination graphs. It is well known that each of the reduced graphs is also chordal.

Observation 5.1. *Let G^* be the reduced subgraph obtained by applying a single step of the Jess and Kees method to a chordal graph G . Then*

1. *A clique of G^* is also a clique of G .*
2. *G^* is a chordal graph.*

Thus, the Jess and Kees method generates a sequence $G_F \equiv G^{(0)}, G^{(1)}, G^{(2)}, \dots, G^{(h)} \equiv \emptyset$ of chordal graphs, where h is the total number of steps in the method. The goal of this section is to demonstrate that there is a simple set of transformations that can be applied to generate a corresponding sequence of clique trees. This will also give an efficient method for identifying new simplicial nodes in the elimination graphs. Throughout this section, we will describe the stepwise transformation of the clique tree representing a chordal graph $G^{(i)}$ into a clique tree representing its successor $G^{(i+1)}$. For simplicity, we shall drop the superscripts and let G denote an arbitrary member (other than the last) of the sequence of chordal graphs and let G^* denote its successor.

Consider the effect of the reduction from G to G^* on the maximal cliques. Let K be a maximal clique in G and define its *residual clique* in G^* by:

$$R(K) = \begin{cases} K & \text{if } K \text{ contains no simplicial nodes in } G, \\ K - \{u\} & \text{where } u \text{ is the eliminated simplicial node of } K \text{ in } G. \end{cases}$$

We call the cliques from which a simplicial node is eliminated, *shrinking cliques*. It is evident that if K is unchanged, it is maximal in G^* as well as G .

All of the action, then, occurs within the shrinking cliques. In particular, $R(K) = K - \{u\}$ need not be a maximal clique in G^* . When this occurs, there is some other maximal clique J of G such that $R(K)$ is contained in a maximal clique $R(J)$ of G^* . We will say that $R(K)$ has been *absorbed* by $R(J)$. We call $R(J)$ an *absorbing clique*. However, it should be borne in mind that no nodes are added to the absorbing clique; we shall show that absorption of a non-maximal clique at most changes the *new, anc* partition of the absorbing clique.

A clique K of G_F is naturally associated with a unique clique $K^{(1)} = R(K)$ in $G^{(1)}$, $K^{(2)} = R(R(K))$ in $G^{(2)}$, \dots , $K^{(i)} = R^{(i)}(K)$ in $G^{(i)}$. For ease of notation, we will associate the same name K with the cliques $K^{(1)}, K^{(2)}, \dots, K^{(i)}$ until such point that $R^{(i)}(K)$ becomes a non-maximal clique, in which case we say that the clique (and the name) K disappear.

An understanding of how the maximal clique structure changes, when one maximal clique shrinks to the point of being absorbed by another clique, is crucial for a fast implementation of the Jess and Kees algorithm. These changes are described in the following theorem.

Theorem 2. *Let G be a chordal graph with clique tree T . Let K be a shrinking clique in G , with residual clique $R(K)$ in G^* .*

- $R(K)$ is non-maximal only if K contained only one simplicial node.
- If $R(K)$ is non-maximal, and K is a leaf clique in T , then $R(K)$ is contained in the residual clique of its parent $P(K)$.
- If $R(K)$ is non-maximal, and K is an interior clique in T , then $R(K)$ is contained in the residual clique of a child C , specifically a child whose ancestor set is largest.

Proof: The first statement follows from the fact that a simplicial node belongs to exactly one maximal clique. If K contained more than one simplicial node, only one was eliminated in the reduction, and so $R(K)$ retains at least one simplicial node from K . This node is not in any other maximal clique, and so $R(K)$ cannot be a subclique of any other maximal clique. Thus a clique can become non-maximal only when its last simplicial node is eliminated.

To prove the second statement, observe that the simplicial nodes of a leaf are precisely its *new* nodes. By Proposition 6, the *anc* nodes of any clique also belong to its parent, and are not simplicial; hence only the *new* nodes in a clique are ever

simplicial. By Theorem 1, the *new* nodes lie only in the leaf and its empty set of descendants; hence all *new* nodes of a leaf are simplicial. Thus a leaf clique becomes non-maximal only when its last *new* node is eliminated. In this case $R(K) = \text{anc}(K)$, which, by Lemma 6, is a subset of the parent clique $P(K)$. As *anc* nodes, these nodes in $R(K)$ were not simplicial in G , and hence they belong to $R(P(K))$ also.

Note that, unlike a leaf, an interior clique need not become non-maximal when its last simplicial node is eliminated, for it has non-simplicial *new* nodes. The third condition in the theorem applies only in those cases where the elimination of the last simplicial node of K causes it to become non-maximal. We prove the result in three steps.

First, we show that if K becomes non-maximal, any clique that absorbs $R(K)$ must be the residual clique of one of the descendants of K . Because K is an interior clique, it has at least one child clique C . The first ancestor node u of C is a *new* node in K and is not simplicial in G . Hence $R(K)$ also contains u . But by Theorem 1, u can belong only to descendants of K , hence only to their residual cliques, and thus only a descendant's residual clique can possibly absorb $R(K)$.

Second, we show that if $R(D)$ absorbs $R(K)$ for some descendant D of K , then $R(C)$ absorbs $R(K)$ for some child C of K . If $R(D)$ absorbs K , we must have $R(K) \subseteq \text{anc}(D)$. Suppose that D is not a child of K . By Theorem 1, any nodes that appear in a clique's ancestor set must appear as *new* nodes in some ancestor and in all cliques on the path in between. All of the nodes in $R(K)$ are *new* nodes in K or one of its ancestors. Hence if they belong to D , they belong also to all cliques lying on the path from D to K in the clique tree, and in particular, to a child of K . Thus, it suffices to consider only the children of K to find out if an absorbing clique exists.

Finally, we show that only a child C with the largest ancestor set has a residual set that can absorb $R(K)$. On the one hand, we know that $\text{anc}(C) \subseteq K$; since none of these nodes is simplicial, we know that for all children C , $\text{anc}(C) \subseteq R(K)$. Hence, $|\text{anc}(C)| \leq |R(K)|$. On the other hand, no node in K or $R(K)$ is found in $\text{new}(C)$; hence $R(K) \subseteq R(C)$ only if $\text{anc}(C) = R(K)$. Thus, such a child C has the largest possible ancestor set among the children of K . Note the stronger result — if any child of maximum ancestor set size can absorb K , so could any other child of maximum ancestor set size. ■

Our identification of non-maximal cliques is central to the Jess and Kees algorithm, since it is only by having cliques become non-maximal that the clique count for any node can decrease, allowing nodes to be identified as simplicial. Corollary 3 gives us a simple test to identify non-maximal cliques at each step of the algorithm.

Corollary 3. *Consider a shrinking clique K whose last simplicial node is eliminated. Then $R(K)$ is non-maximal if and only if either $|new(R(K))| = 0$ or a child C of K with the largest ancestor size satisfies $|anc(C)| = |R(K)|$.*

Proof: First, observe that the two conditions are mutually exclusive: the first applies to a leaf clique and the second to an interior clique. If the first condition holds, K must be a leaf of T , since we showed in the proof above that an interior clique must have a non-simplicial *new* node. Conversely, if the first condition fails, then K was not a leaf because all *new* nodes of leaves are simplicial, but at least one of K 's *new* nodes was not simplicial.

The proof is then a straight-forward extension of the proof of Theorem 2. If $R(K)$ is non-maximal, and K is a leaf, then we have already shown that $R(K) = anc(K)$, and hence $new(R(K)) = \emptyset$. If $R(K)$ is non-maximal and K is an interior clique, then again from the proof of Theorem 2, $R(K) = anc(C)$.

Conversely, if $|new(R(K))| = 0$, then K is a leaf and $R(K) = anc(K) \subseteq P(K)$. Then $R(K)$ is non-maximal. If the second condition applies, we have $anc(C) = R(K)$, since we know that $anc(C) \subseteq R(K)$. Hence $R(K) \subseteq R(C)$, and again $R(K)$ is not maximal. ■

The next goal then is to build a clique tree for G^* . To avoid a possible ambiguity that arises from eliminating a maximum independent set of simplicial nodes simultaneously, we arbitrarily order the simplicial nodes that are eliminated from G at this step as u_1, u_2, \dots, u_t . We will build the clique tree for G^* from the clique tree for G in t minor steps, where the i -th step removes u_i from $G - \{u_1, \dots, u_{i-1}\}$ and properly alters the clique tree. Note that in a given minor step in which a node u_i is removed, u_i belongs to exactly one clique K ; for all other cliques \tilde{K} , it follows that $R(\tilde{K}) = \tilde{K}$. The rules for transforming the tree at each step are described in the following Theorem.

Theorem 4. *Let G be a chordal graph with a clique tree T . Let u be a simplicial node, belonging to a maximal clique K , that is eliminated from G . Then a clique tree for $G - \{u\}$ can be constructed by the following rules:*

- If $R(K)$ is maximal, make no changes.
- Otherwise,
 - if K is a leaf of T , prune K from T .
 - if K is an interior clique of T , let C be a child with the largest ancestor set among the children of K . Then:
 - * assign all nodes in $new(K)$ to $new(C)$, removing them from $anc(C)$

- * attach C as a child of $P(K)$, with $first_anc(C)$ reset to $first_anc(K)$
- * attach children of K other than C as children of C
- * remove K from the tree.

Proof: We must show that the changes preserve a correct *new*, *anc*, *first_anc* labeling of each clique. Note that u appeared only in K , and hence all other cliques remain the same after the elimination of u .

The result is trivial when $R(K)$ remains maximal. The only change made to the tree is to remove a node from $new(K)$ that appeared only in K . Thus, there are no structural changes in the clique tree.

There are two cases to consider when $R(K)$ is not maximal, corresponding to whether K was a leaf of the clique tree or an interior clique. If K was a leaf, u was the last node in $new(K)$, which now becomes empty. This node appeared only in K , so the partition of nodes elsewhere is unaffected by removing K from the tree.

In the case of an interior clique, we need to check that the revised partition satisfies the required conditions for a parent and its child. For ease of notation, let $G' \equiv G - \{u\}$. We use primed quantities to refer to updated variables in G' . Note that when K is an interior clique absorbed by a child C , $R(K) = anc(C)$. Also, $anc(C)$ is an ordered subset of C consisting of all nodes numbered higher than or equal to C 's first ancestor. By our rules,

$$\begin{aligned} new'(C) &= new(C) \cup (new(K) - \{u\}), \\ anc'(C) &= anc(C) - (new(K) - \{u\}) = anc(K). \end{aligned}$$

Thus our rules update C by moving nodes numbered lower than the first ancestor of K from the *anc* set of C to the *new* set of C . We now show that the revised partition $new'(C)$, $anc'(C)$ of C is correct, and then that the parent-child relationships in the new clique tree are correct.

If a node w is included in $new'(C)$ by these rules, either it was in $new(C)$ or in $new(R(K))$. In the former case, $Madj(w) \subseteq C$ before the elimination of u , and this relationship is unaffected by the elimination of u . In the latter case, $Madj(w) \subseteq K$ before the elimination of u implies that $Madj'(w) \subseteq R(K) = anc(C) \subset C$. In both cases, by Proposition 4, Algorithm 3.1 applied to the graph G' would mark w as a *new* node in C .

Similarly, if w is included in $anc'(C)$, then $w \in anc(K)$. Therefore, by Proposition 4, $Madj(w) \not\subseteq K$, and thus $Madj'(w) \not\subseteq R(K)$. Each node in $Madj(w)$ is numbered higher than w , which in turn is numbered higher than each node in $new(C)$ since $w \in anc(C)$.

Therefore,

$$Madj'(w) \not\subseteq R(K) \cup new(C) = anc(C) \cup new(C) = C.$$

Thus Algorithm 3.1 applied to G' would not mark w as a *new* node in C .

$P(K)$ is the correct choice as the parent of C because $first_anc'(C) = first_anc(K)$, a node in $new(P(K))$. To demonstrate that the other children of K have been properly reassigned in T' , consider any such child J . The eliminated node u was simplicial, and therefore did not appear in $anc(J)$. Hence $anc'(J) = anc(J) \subseteq R(K) = C$. The node $first_anc(J)$ was in $new(K)$ and so becomes a member of $new'(C)$. C is then the correct choice for the parent of J . ■

When a clique loses its last simplicial node, we need to check if it is non-maximal. Clearly, maintaining a count of the number of *new* nodes suffices for leaf cliques. For an interior clique we need to find a child that can absorb it, without searching through all of its children. This is easy to do in the initial clique tree, since the children of a clique are sorted by ancestor set size, and the first child can absorb its non-maximal parent. However, this becomes a subtle issue for a clique tree at some intermediate stage in the Jess and Kees method, since a clique may absorb its parent and acquire new children, and then its children may no longer be ordered in decreasing ancestor size. Thus it might appear that we have to re-sort the child cliques every time a clique absorbs new children.

The next two results show that such re-sorting is not necessary. The initial child ordering by ancestor size may be destroyed by interior clique absorptions, but a careful treatment of the child lists can maintain an ordering in which only the first child needs to be considered as a candidate for absorbing a parent clique. These results will turn out to be crucial in obtaining a linear time bound for our implementation of the Jess and Kees method.

Proposition 7. *Let C_1 and C_2 be siblings in the clique tree for G_F , with $|anc(C_1)| \geq |anc(C_2)|$. Then $|anc(C_1)| \geq |anc(C_2)|$ holds at all later stages of the method as long as C_1 and C_2 continue to be siblings.*

Proof: Let $|anc(C_1)| \geq |anc(C_2)|$ in the initial clique tree. No updating operations increase the size of the ancestor sets, and ancestor nodes are never simplicial, so a change in this relationship can be caused only by nodes being transferred from $anc(C_1)$ to $new(C_1)$ or by the disappearance of either C_1 or C_2 . In the latter case, they are trivially no longer siblings. The former case is nearly as trivial because it can only occur if C_1 absorbs its parent, in which case all of its siblings become its children. Obviously then C_2 will no longer be a sibling of C_1 . ■

The gist of Proposition 7 is that the original children of K remain in the same relative order even if some of the children are eliminated from the tree. Thus, the elimination of a leaf of the clique tree requires no changes in how its siblings are listed. When an interior clique is absorbed, a simple ordering of the revised child lists will enable us to continue testing only the first child for absorbing its parent clique.

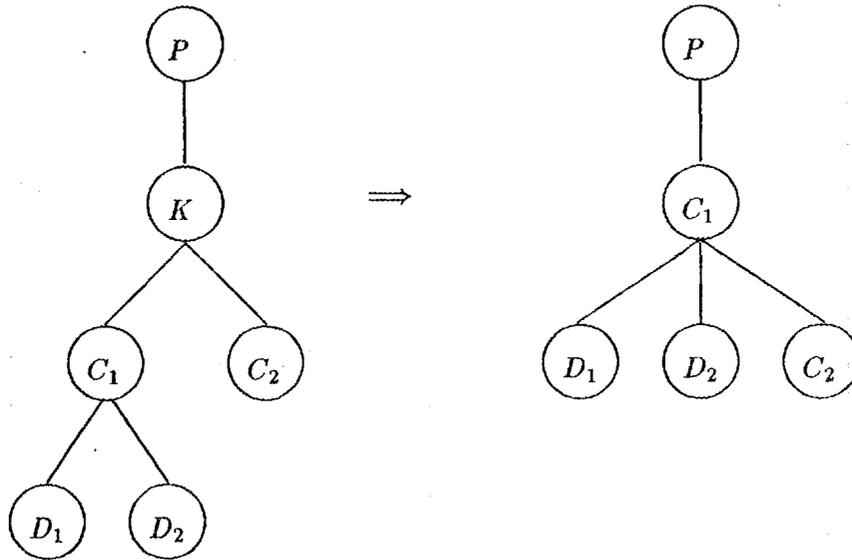


Figure 3: Updating the clique tree when an interior clique K is absorbed by its first child C_1 .

Our rule for arranging the revised child lists when an interior clique is absorbed is as follows. Let C_1 be the first child which absorbs its parent clique K . Let the other children of K be ordered C_2, \dots, C_i , and let the children of C_1 be ordered D_1, \dots, D_j . After the update of the tree, arrange the children of C_1 in the order $D_1, \dots, D_j, C_2, \dots, C_i$. This requires only the minimal cost of appending one list of children to the end of another list. Figure 3 illustrates this rule. The next Theorem shows that this rule can be applied whenever an interior clique is absorbed during elimination.

Theorem 5. *Assume that the lists of children in the clique tree are initially sorted by decreasing ancestor set size, and that the child lists are revised by the above rule*

when an interior clique is absorbed. Then, at any future step in the elimination, a non-maximal interior clique can be absorbed by its first child.

Proof: We prove that the rule for arranging child lists when an interior clique is absorbed preserves the property that a non-maximal clique can be absorbed by its first child. First, we introduce the concept of a *cohort*: we group the children of a maximal clique into ordered sets called cohorts.

We define cohorts inductively. If K is a maximal clique in the original clique tree, its children form a *cohort* \mathcal{H} . When K becomes non-maximal and is absorbed by some child C_l , the cohort structures change as follows: Let P be the parent of K , the children of K be ordered $C_1, \dots, C_l, \dots, C_i$, and let the children of C_l be ordered D_1, \dots, D_j . In the clique tree C_l takes the place of K as a child of P ; hence the clique C_l is removed from the cohort it belonged to, and joins the cohort that K belonged to. The cliques $C_1, \dots, C_{l-1}, C_{l+1}, \dots, C_i$ become new children of C_l , listed after D_1, \dots, D_j ; if the latter cliques form q cohorts ordered as $\mathcal{H}_1, \dots, \mathcal{H}_q$, and the former cliques form $r - q$ cohorts ordered as $\mathcal{H}_{q+1}, \dots, \mathcal{H}_r$, then after the tree update, the children of C_l form r cohorts ordered as $\mathcal{H}_1, \dots, \mathcal{H}_q, \mathcal{H}_{q+1}, \dots, \mathcal{H}_r$. Note that the relative ordering of the cohorts within each group of children is preserved. We will prove that the absorbing child C_l can always be taken to be the first child C_1 .

We use the concept of a cohort only in this proof; in our algorithms we do not need to maintain the cohort structures at parent cliques.

Let J be a maximal clique in the clique tree at some intermediate stage in the elimination, with its children grouped into *cohorts* $\mathcal{H}_1, \dots, \mathcal{H}_t$. Except for the uneliminated children of J from the original clique tree, other cohorts are acquired when J absorbs a parent clique.

We claim that our rule for merging child lists preserves the following two properties of the cohort structure at any clique J :

1. within each cohort \mathcal{H}_s , the cliques are listed in decreasing order of ancestor size.
2. each child in a cohort \mathcal{H}_s contains some node in $new(J)$ that is found in no clique in the cohorts $\mathcal{H}_{s+1}, \dots, \mathcal{H}_t$.

We prove the claim by induction on the number of interior clique absorptions. It holds trivially before the first such absorption because each clique has only one cohort, its original children, which are ordered by ancestor size. For the inductive step, assume the properties hold before the absorption of a non-maximal clique K by its first child C_1 . As before, denote the parent of K by P , the other children of K by C_2, \dots, C_i , and the children of C_1 before the clique tree update by D_1, \dots, D_j . Let $R(K)$ denote

the set of nodes in K when it is absorbed, and denote the ancestor and *new* sets of C_1 after the update by $anc'(C_1)$, and $new'(C_1)$, respectively; note that the former set is equal to $anc(R(K))$, and similarly the latter set is $new(C_1) \cup new(R(K))$. Assume also that C_1 's children D_1, \dots, D_j before the absorption form q cohorts, $\mathcal{H}_1, \dots, \mathcal{H}_q$, and that its new children C_2, \dots, C_i form $r - q$ cohorts, $\mathcal{H}_{q+1}, \dots, \mathcal{H}_r$.

Note that the only cliques whose child lists are modified by the absorption of K are the two cliques P and C_1 . It is only for these two cliques that we need to prove that the properties hold.

The child list for P is modified only by replacing K with C_1 , and the relative order of the cohorts is unchanged. Also, $anc'(C_1) = anc(R(K))$, so the ancestor sets of the children of P are unchanged. Since only the relative ordering of the cohorts and the ancestor sets are addressed by both properties, they continue to hold for P .

It remains to examine the children of C_1 . The first property holds trivially, since within a cohort we do not change the relative order of the cliques. Hence we prove the second property.

Consider the situation before the absorption of K by C_1 . For $k = 1, \dots, j$, let the first ancestor of D_k be v_k . These nodes are not necessarily distinct; let $V = \bigcup_{k=1}^j \{v_k\}$ be the corresponding set of nodes. Each node in V belongs to $new(C_1)$, but not to C_2, \dots, C_i , since the latter cliques are siblings and not descendants of C_1 . This result continues to hold after the clique tree update, when these cliques become new children of C_1 .

Thus, we have shown the second property with respect to the two groups of cohorts, $\mathcal{H}_1, \dots, \mathcal{H}_q$, and $\mathcal{H}_{q+1}, \dots, \mathcal{H}_r$. By the induction hypothesis, this property held within each group of cohorts before the update. It holds after the update for the first group trivially. It holds for the second group because $new'(C_1)$ contains all nodes in $new(R(K))$ included in C_2, \dots, C_i . This completes the proof of the claim.

The consequence of the second property for the Jess and Kees method is the following: For each child D in its first cohort and for every child C in later cohorts, there exists some node v in both $new(J)$ and D , but not in C . Thus the child clique C cannot absorb J until this node v has been eliminated. But before v is eliminated, it must first become simplicial, which requires that D become non-maximal and be deleted from the clique tree. Thus, only a child belonging to the first cohort can possibly absorb J .

Within each cohort, the children are listed in decreasing order of ancestor set size; in particular, this is true for the first cohort of J . From Theorem 2 and from Proposition 7, only a child of largest ancestor set size in the first cohort can absorb J . The result is then that the structure induced by our child list merging rule ensures that only the first child needs to be tested as a candidate for absorption. ■

We conclude this section with a discussion of the operations of downdating clique memberships counts and of finding new simplicial nodes. In both cases the proper view simplifies the data structures required to make the identification.

The count of clique membership for a node changes only when a clique containing it is absorbed by another clique. The rule for changing membership counts is: When a clique $R(K)$ is absorbed, reduce the clique count by 1 for each node in $R(K)$. In the case of a leaf clique, this set of nodes is $anc(K)$. When an interior clique is absorbed, this set is $anc(C)$, where C is the absorbing child. In both cases, the set of nodes is a contiguous trailing segment of the list of row subscripts for a representative column of L , and can be obtained easily from the array of row subscripts.

Which nodes can be simplicial at the next step of Jess and Kees? Which cliques will be the shrinking cliques at the next step? Any node that was simplicial and was not eliminated remains simplicial. Thus, any shrinking clique whose set of simplicial nodes was not exhausted remains a shrinking clique. The only nodes that can become newly simplicial are those whose clique membership counts become one when downdated. Such nodes belong to a clique that became non-maximal, and also to the clique that absorbed the non-maximal clique. Hence only a maximal clique that absorbed another clique can possibly become a new shrinking clique. Thus we have proved the following corollary:

Corollary 6. *Let G^* be the graph resulting from elimination of an independent set of simplicial nodes from G . Then:*

- *a node can be simplicial in G^* only if it was simplicial in G or it lies in a clique that absorbed another clique during this elimination*
- *a clique of G^* can be a shrinking clique only if it was a shrinking clique for G or it absorbed some other clique during this elimination.*

6. A Parallel Reordering Algorithm

We are now ready to describe our parallel reordering algorithm. We assume that the initial clique tree T has been constructed, that the maximal cliques are numbered from 1 to m , and that the children of each maximal clique are ordered in decreasing ancestor set size. We represent each maximal clique K in the clique tree by the list of the nodes in $anc(K)$. Our algorithm does *not* require the list of nodes in $new(K)$ —it will suffice to maintain the number $|new(K)|$ of such nodes. As noted previously, the subscript lists for the Cholesky factor L suffice as the list of *anc* nodes.

In addition, we assume that we are given for each node v , $clique_count(v)$, the number of maximal cliques to which v belongs, and $new_in_clique(v)$, the maximal clique in which v is a *new* node. We also require the list \mathcal{L} of shrinking cliques, and the list of simplicial nodes $S(K)$ in each shrinking clique K . The list \mathcal{L} is maintained as a linked list, and each list $S(K)$ is maintained as a queue. $R(K)$ will denote the residual set of nodes immediately after the removal of a simplicial node from K .

To update the clique tree when a clique becomes non-maximal, we may require the parent of a leaf clique, or the parent and children of an interior clique. Hence we represent the clique tree in terms of both the parent relationship and the child relationship. The latter representation is explicitly maintained by means of the *first_child* and *right_sibling* lists. The *right_sibling* list is maintained as a doubly linked list to permit constant time deletions of any clique in the list. We do not represent the parent relationship explicitly by means of the *parent* vector; it does not seem possible to update the *parent* vector explicitly and obtain a linear algorithm. Instead, we maintain the parent relationship implicitly by means of the vectors *first_anc* and *new_in_clique*. If $first_anc(K) = v$, then v is a *new* node in K 's parent P . Hence $new_in_clique(v) = P$. Thus the *first_anc* and *new_in_clique* vectors enable us to identify the parent clique in constant time.

At the outset, all the required data structures can be computed in $\mathcal{O}(n + q)$ time from Algorithm 3.1. In addition, we can sort the children of all the maximal cliques with a careful bucket sort in $\mathcal{O}(n)$ time.

Algorithm 6.1 computes the parallel reordering from the clique tree, numbering nodes from 1 to n . Each iteration of the **while** loop corresponds to one step of the Jess and Kees method. In each step, one simplicial node is eliminated from each shrinking clique in \mathcal{L} . After the elimination of a simplicial node v from a shrinking clique K , the clique tree and other data structures are updated as necessary. The list \mathcal{L}_{new} is the list of new shrinking cliques. At the end of each step, these new shrinking cliques are added to the existing list of shrinking cliques.

An auxiliary procedure *Downdate*, is used to decrement $clique_count(v)$ for each node v in a clique K that becomes non-maximal, to update the queue $S(J)$ of simplicial nodes in the clique J that absorbs K , and to add, if necessary, the absorbing clique J to the list \mathcal{L}_{new} . This procedure is shown separately.

The operations associated with a non-maximal interior clique K require some comment. When such a clique K is deleted, the set of nodes in it, $R(K)$, is precisely the set $anc(C)$ for its first child C . After the absorption of K , $first_anc(C)$ needs to be reset to $first_anc(K)$, and the partition $new(C)$, $anc(C)$ needs to be updated. By our rules, all nodes in $new(R(K))$ are removed from $anc(C)$ and added to $new(C)$. We now discuss

Algorithm 6.1.

```

Make  $\mathcal{L}_{new}$  the empty list
while  $\mathcal{L}$  is not empty do
  for each  $K \in \mathcal{L}$  do
    number the first simplicial node  $v \in S(K)$ .
    remove  $v$  from  $S(K)$  and decrement  $|new(K)|$ 
    if  $S(K)$  is empty then
      delete  $K$  from  $\mathcal{L}$ 
      if  $K$  is a leaf then /* $R(K) = anc(K)$ */
        delete  $K$  from  $T$ 
        Downdate( $anc(K)$ ,  $P(K)$ )
      else
         $C := first\_child(K)$ 
        if  $|R(K)| = |anc(C)|$  then /* $R(K) = anc(C)$ */
           $v := first\_anc(C)$ 
          for  $i := 1$  to  $|new(K)|$  do
             $new\_in\_clique(v) := C$ 
             $v := next\ node\ in\ anc(C)$ 
          endfor
          Downdate( $anc(C)$ ,  $C$ )
           $first\_anc(C) := v$ 
          update  $|new(C)|$  and  $|anc(C)|$ 
          delete  $K$  from  $T$  and replace  $K$  by  $C$  as child of  $P(K)$ 
          append other children of  $K$  to end of children list of  $C$ 
        endif
      endif
    endif
  endfor
  Add the shrinking cliques in  $\mathcal{L}_{new}$  to  $\mathcal{L}$ 
  Make  $\mathcal{L}_{new}$  the empty list
endwhile

```

how this is accomplished, since we do not explicitly maintain the set $new(R(K))$, but only its size.

Since we assume that the nodes in the maximal cliques are ordered by a symbolic factorization procedure, the set $anc(C)$ is a contiguous list of nodes, listed in increasing order of node numbers. Since $R(K) = anc(C)$, and nodes in $new(R(K))$ are numbered lower than nodes in $anc(R(K))$, the nodes in $new(R(K))$ are the first $|new(R(K))|$

```

procedure Downdate( $U, J$ )
{downdate  $clique\_count(u)$  for the set of nodes  $U$  in an absorbing clique  $J$ ,
and identify new simplicial nodes and new shrinking cliques}
for each  $u \in U$  do
     $clique\_count(u) := clique\_count(u) - 1$ 
    if  $clique\_count(u) = 1$  then
        /* $u$  is simplicial*/
        if  $S(J)$  is empty then
            set the queue  $S(J) := \{u\}$ 
            Add  $J$  to the list  $\mathcal{L}_{new}$ 
        else
            Add  $u$  to the queue  $S(J)$ 
        endif
    endif
endfor

```

nodes in $anc(C)$. For these nodes, we reset $new_in_clique(v)$ to be the clique C , and then $first_anc(C)$ is the node immediately following these nodes in $anc(C)$. We perform the Downdate operation first, and then update $first_anc(C)$, and implicitly, the set $anc(C)$.

In the remainder of this section we verify our claims regarding the complexity of the algorithm. Clearly, in the algorithm as written, the only non-constant work in deleting a non-maximal clique is the $\mathcal{O}(|R(K)|)$ operations in the call to Downdate and resetting the new_in_clique values. Let $\overline{R(K)}$ be the residual clique of K when it becomes non-maximal. Each call of downdate requires $\mathcal{O}(|\overline{R(K)}|)$ time, and hence the total cost of all calls of Downdate is

$$\sum_{K \in T} \mathcal{O}(|\overline{R(K)}|) \leq \sum_{K \in T} \mathcal{O}(|K|) = \mathcal{O}(q).$$

We now compute the complexity of the rest of the algorithm. The cost of eliminating simplicial nodes, without considering clique tree updates, is $\mathcal{O}(n)$. There are m cliques initially, so the constant work of removing a clique takes $\mathcal{O}(m) \leq \mathcal{O}(n)$ time overall. Thus the algorithm has time complexity $\mathcal{O}(n + q)$.

Table 1: Test Matrices from the Harwell-Boeing Sparse Matrix Collection

key	description	order
BCSPWR09	Western US power grid	1723
BCSPWR10	Eastern US power grid	5300
BCSSTK08	TV Studio structure	1074
BCSSTK13	Fluid Flow Stiffness Matrix	2003
BCSSTM13	Fluid Flow Mass Matrix	2003
BLCKHOLE	Geodesic Dome	2132
CAN 1072	Aircraft structure	1072
DWT 2680	Naval destroyer structure	2680
LSHP 3466	L-shaped regular grid	3466
GR 40x40	40 by 40 square grid	1600
GR 80x80	80 by 80 square grid	6400

7. Empirical Results

In this section we present experimental results on the parallel reordering algorithm, and compare the performance of our algorithm with the Liu and Mirzaian algorithm and Liu's Composite Rotations heuristic.

We have written Fortran 77 codes for Algorithms 3.1 and 6.1. An additional procedure was written to create the data structures required by the latter algorithm. Each maximal clique K was represented by the nodes in $anc(K)$; this set of nodes is already represented in the subscript lists for L , if we maintain a pointer to the first ancestor node of each maximal clique. The subscript lists for L were generated by a symbolic factorization of A .

We used eleven problems from the Harwell-Boeing test set [4,3] to test our algorithm. These are described in Table 1. The storage statistics for the Cholesky factorizations of these matrices, after ordering with the multiple minimum degree heuristic [11], are given in Table 2. These statistics give the primary terms in the complexities of each of the three algorithms. Liu's heuristic requires time almost linear in the number of nonzeros in A ; the Liu and Mirzaian algorithm runs in time linear in the number of nonzeros in $L + L^T$; and our algorithm is linear in the number of elements in the mass-elimination compressed subscript array. It is clear from Table 2 that the size of the representation of the original matrix A and the size of the compressed subscript representation for L are similar, and much smaller than the size of L . Table 2 also lists the number of maximal cliques in the filled graph for each problem.

Our experiments were performed on a Sun 3/260 workstation running Sun Unix 3.3, and we used the f77 compiler with the optimization turned on. Because of the

Table 2: Fill Statistics from the Multiple Minimum Degree Ordering, Subscript Compression only by Mass Elimination

key	order	number of nonzero entries in lower triangle of			number of maximal cliques
		A	compressed subscripts	L	
BCSPWR09	1723	2394	3995	4592	1621
BCSPWR10	5300	8271	16843	22764	4846
BCSSTK08	1074	5943	10845	29973	756
BCSSTK13	2003	40940	28027	269668	597
BCSSTM13	2003	9970	6972	43491	1540
BLCKHOLE	2132	6370	13053	51012	1150
CAN 1072	1072	5686	7091	19332	620
DWT 2680	2680	11173	18410	50072	1550
LSHP 3466	3466	10215	21270	83116	1845
GR 40x40	1600	12324	10299	33304	845
GR 80x80	6400	50244	44222	183301	3285

coarse granularity of the timing subroutines on this machine, we obtain the times by averaging over several hundred repetitions. In Table 3, we report the CPU times required by the four steps in computing the parallel ordering. First is the fill-reducing minimum degree ordering step, second is the symbolic factorization step, next the procedure that computes the initial clique tree and other data structures (setup costs), and finally Algorithm 6.1. These results show that the minimum degree ordering time dominates the other three steps. Also note that the time required to create the initial clique tree and the required data structures is comparable to the time taken by the parallel reordering algorithm. Both Algorithm 3.1 and Algorithm 6.1 are $\mathcal{O}(n + q)$ in complexity, and hence these results are in accord with our complexity analyses.

The second most expensive step is the symbolic factorization. The procedure we used is a modification of the Fortran program `smbfct` in [7]. We made several modifications to this code to take greater advantage of mass elimination compression. We removed the limited capability for finding accidental compression from the procedure. This has the advantage of exhibiting precisely the size of the representation of the maximal cliques of the filled graph. It also results in a faster symbolic factorization, running 8% faster, while increasing storage by 0.4% on average. In addition, to improve its speed we rewrote the initial processing that detects mass elimination before creating the subscript lists speed. As a result our symbolic factorization is somewhat faster than `smbfct`.

Table 3: Time to compute a parallel ordering (CPU Seconds) on a Sun 3/260

key	min degree ordering	symbolic factorization	setup costs	Algorithm 6.1
BCSPWR09	0.48	0.08	0.08	0.07
BCSPWR10	1.96	0.31	0.26	0.23
BCSSTK08	2.78	0.12	0.06	0.05
BCSSTK13	4.56	0.44	0.11	0.10
BCSSTM13	1.03	0.13	0.08	0.04
BLCKHOLE	0.69	0.15	0.09	0.08
CAN 1072	0.69	0.09	0.05	0.04
DWT 2680	1.78	0.23	0.13	0.11
LSHP 3466	1.12	0.25	0.15	0.13
GR 40x40	0.51	0.13	0.07	0.06
GR 80x80	2.06	0.53	0.29	0.25

Codes for Liu's heuristic and for the Liu and Mirzaian algorithm were not available to us for direct comparison. However, Liu has published timing results for his algorithms applied to the problems in Table 1, using a SUN 3/50 workstation. This computer is closely related to the SUN 3/260 we used, but it is not as powerful. However, Liu has used his multiple minimum degree ordering code [11] for the first step, and we also used this code to find the fill-reducing ordering. The common minimum degree ordering code consistently ran approximately 4 times faster on our workstation. To make our results comparable to Liu's results, we normalize the statistics published by Liu [13] to a cost relative to that of the common minimum degree ordering. For all three approaches, we report the ratio of the CPU time required for the postprocessing to the CPU time for the minimum degree ordering on the same machine. In our case the postprocessing requires a symbolic factorization, computation of the initial clique tree and other data structures, and Algorithm 6.1, all of which together we refer to as the Clique Tree Jess and Kees algorithm. The total time for all of these steps in the Clique Tree Jess and Kees algorithm is reported in Table 4.

From the relative performances in Table 4, it is clear that the Liu and Mirzaian algorithm is much more expensive than the other two. Between our algorithm and Liu's Composite Rotations algorithm, we see a significant difference on only three of the eleven problems. The difference in performance on the remaining eight problems is smaller than the variation in relative times we have seen with our own code at various levels of compiler optimization.

The three problems with extremal behavior distinguish between the codes in a predictable way. The first two, BCSPWR09 and BCSPWR10, are electric power problems,

Table 4: Performance relative to Minimum Degree Ordering Time

key	comparative performance		
	Composite Rotations	Clique Tree Jess and Kees	Liu & Mirzaian Jess and Kees
BCSPWR09	0.21	0.48	0.54
BCSPWR10	0.18	0.41	0.55
BCSSTK08	0.06	0.08	0.53
BCSSTK13	0.26	0.14	1.74
BCSSTM13	0.27	0.24	1.30
BLCKHOLE	0.38	0.47	1.93
CAN 1072	0.19	0.26	0.98
DWT 2680	0.21	0.26	0.90
LSHP 3466	0.42	0.48	2.04
GR 40x40	0.38	0.50	1.66
GR 80x80	0.49	0.52	2.42

a class of problems where very low fill is typical and the number of maximal cliques is close to n . For these problems essentially all operations are $\mathcal{O}(n)$, and the use of cliques makes only relatively small improvements on the Liu and Mirzaian implementation of Jess and Kees. The third problem, BCSSTK13, is at the other extreme. This is a large finite element model with a number of degrees of freedom associated with each physical node. Mass elimination compression has an enormous effect, similar to what we usually see on much larger problems, which means that the maximal clique representation is very compact. The compressed subscripts lists are smaller than the representation of the original matrix A ; the complexity bounds correctly predict that our algorithm is faster than Liu's heuristic.

After the parallel ordering is computed, we will require the structure of the reordered L to compute the numeric factorization. The equivalence of the subscript lists and the maximal cliques used to create the initial clique trees for the Clique Tree Jess and Kees algorithm may serve to further advantage. The maximal cliques are the same for the parallel ordering as for the sequential ordering; the uneliminated nodes in a clique K from which a node u is eliminated in Algorithm 6.1 compose the subscript list for u . A symbolic factorization for the parallel ordering based on this observation has been written, and it proved to be somewhat faster than the modified `smfct` procedure. In contrast, the tree rotations algorithm must be followed by a general symbolic factorization step. Since symbolic factorization accounts for about half the post processing time in our algorithm, the comparison of the relative performances of

these algorithms appears favorable to the tree rotations algorithm. But it remains to be seen whether the completed symbolic factorization provides an advantage in practice.

8. Concluding Observations

The concept of a clique tree has previously appeared in the graph theory literature and in the context of acyclic relational database schemes. Buneman [2], Gavril [6], and Walter [21] have considered the clique tree representation of chordal graphs, and Golombic [8] contains a good discussion of several properties of chordal graphs. In the context of database theory, an acyclic hypergraph captures the relations in acyclic database schemes, and a clique tree can be used to represent this hypergraph. Beeri et al. [1] and Tarjan and Yannakakis [20] have considered the clique tree in this context and call it a join tree. In his thesis, Peyton [16] discusses some applications of clique trees to sparse linear systems, including the design of a multifrontal Cholesky factorization algorithm based on the clique tree.

The method we use to compute a clique tree differs from the approaches employed in the graph theory literature and in database theory since we have more information about the chordal filled graph that we can take advantage of. At the outset, we have a perfect elimination order for the filled graph from the fill-reducing ordering, and also a compact representation of the maximal cliques from the compressed row subscripts available from the symbolic factorization step. By using this information, Algorithm 3.1 generates a clique tree efficiently with regard to both storage and time. Indeed, if we do not compute the *clique_count* information (which is not necessary to generate the clique tree), this algorithm computes the clique tree in $\mathcal{O}(n)$ time.

Algorithms that simultaneously compute a perfect elimination order and a clique tree cannot be as efficient. Tarjan and Yannakakis [20] first renumber the nodes by a generalized maximum cardinality search of the hypergraph and then compute a clique tree from this ordering in $\mathcal{O}(n+q)$ time. Their algorithm requires node-clique incidence lists in addition to the node lists of the maximal cliques. Peyton [16] has designed an $\mathcal{O}(n + \eta(F))$ algorithm to compute simultaneously the maximum cardinality ordering of a chordal graph and a clique tree when the graph is represented by adjacency lists.

Note that we define a clique tree algorithmically, by means of Algorithm 3.1. There is a significant difference between our definition of a clique tree and the definitions employed in other contexts. We require the first ancestor node of a clique to be a *new* node in its parent clique. Previous definitions have required a clique tree to satisfy only the last two properties in Proposition 6. With our more restrictive definition, we acquire the property that the parent P is the *only* ancestor of a maximal clique K that contains all nodes in the set $anc(K)$; other ancestors of K can contain only proper

subsets of the nodes in $anc(K)$. This property turned out to be useful in updating the clique tree by local changes during elimination, and is at the heart of the linear time bounds in our parallel reordering algorithm.

The clique tree representation of the filled graph has consequences in several other sparse matrix problems as well. The clique tree helps to clarify the notion of a super-node in vectorized numeric factorization. It makes possible the design of a multifrontal factorization in which the fronts are precisely the maximal cliques, thereby requiring the minimum number of frontal elimination steps. A carefully designed clique tree multifrontal algorithm may have smaller working storage requirements than a multifrontal algorithm based on an elimination tree. The clique tree aids in the understanding of how elimination tree restructuring algorithms change the height of an elimination tree. It is useful in the context of sparse orthogonal factorization algorithms also. The notion of a 'best' clique tree in a particular context needs careful study as well. We intend to explore several of these issues in future work.

Acknowledgements. Preliminary versions of some results in this paper were included in [10] and [17]. Alex Pothen's work on this paper was begun while visiting Oak Ridge National Laboratory and The University of Tennessee, Knoxville, during their special year in numerical linear algebra. He thanks Alan George for the invitation to offer a graduate course on Sparse Matrix Algorithms, which provided the stimulus for this research; he also thanks other colleagues at both institutions for their hospitality and encouragement.

9. References

- [1] C. BEERI, R. FAGIN, D. MAIER, AND M. YANNAKAKIS, *On the desirability of acyclic database schemes*, J. Assoc. Comput., 30 (1983), pp. 479-513.
- [2] P. BUNEMAN, *A characterization of rigid circuit graphs*, Discrete Math., 9 (1974), pp. 205-212.
- [3] I. DUFF, R. GRIMES, AND J. LEWIS, *Sparse Matrix Test Problems*, Tech. Rep. CSS 191, Harwell Laboratory, Didcot, Oxon, England, 1987.
- [4] I. DUFF, R. GRIMES, J. LEWIS, AND W. POOLE, JR., *Sparse matrix test problems*, SIGNUM Newsletter, 17,2 (1982), p. 22.
- [5] I. S. DUFF AND J. K. REID, *A note on the work involved in no-fill sparse matrix factorization*, IMA J. of Numer. Anal., 3 (1983), pp. 37-40.

- [6] F. GAVRIL, *The intersection graphs of subtrees are exactly the chordal graphs*, J. Comb. Theory, Ser. B, 16 (1974), pp. 47–56.
- [7] A. GEORGE AND J. W. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, Englewood Cliffs, NJ, 1981.
- [8] M. GOLOMBIC, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
- [9] J. JESS AND H. KEES, *A data structure for parallel L/U decomposition*, IEEE Trans. Comput., C-31 (1982), pp. 231–239.
- [10] J. LEWIS AND B. PEYTON, *A fast implementation of the Jess and Kees Algorithm*, Tech. Rep. ETA-TR-90, Boeing Computer Services, Seattle, WA, 1988.
- [11] J. W. LIU, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141–153.
- [12] ———, *Computational models and task scheduling for parallel sparse Cholesky factorization*, Parallel Computing, 3 (1986), pp. 327–342.
- [13] ———, *Reordering Sparse Matrices for Parallel Elimination*, Tech. Rep. CS-87-01, York University, North York, Ontario, Canada, 1987.
- [14] ———, *The Role of Elimination Trees in Sparse Factorization*, Tech. Rep. CS-87-12, York University, North York, Ontario, Canada, 1987.
- [15] J. W. LIU AND A. MIRZAIAN, *A Linear Reordering Algorithm for Parallel Pivoting of Chordal Graphs*, Tech. Rep. CS-87-02, York University, North York, Ontario, Canada, 1987. Also SIAM J. Disc. Math., to appear.
- [16] B. PEYTON, *Some Applications of Clique Trees to the Solution of Sparse Linear Systems*, PhD thesis, Clemson University, Clemson, SC, 1986.
- [17] A. POTHEN, *Simplicial Cliques, Shortest Elimination Trees, and Supernodes in Sparse Cholesky Factorization*, Tech. Rep. CS-88-13, Computer Science, Pennsylvania State University, University Park, PA, April 1988.
- [18] ———, *The Complexity of Optimal Elimination Trees*, Tech. Rep. CS-88-16, Computer Science, Pennsylvania State University, University Park, PA, April 1988.
- [19] D. ROSE, *A graph-theoretic study of the numerical solution of sparse positive definite systems of equations*, in Graph Theory and Computing, R. Read, ed., Academic Press, New York, 1972, pp. 183–217.

- [20] R. TARJAN AND M. YANNAKAKIS, *Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, SIAM J. Comput., 13 (1984), pp. 566–579.
- [21] J. R. WALTER, *Representations of chordal graphs as subtrees of a tree*, J. Graph Theory, 2 (1978), pp. 265–267.

ORNL/TM-11040

INTERNAL DISTRIBUTION

- | | |
|--------------------------|--------------------------------------|
| 1. B. R. Appleton | 26-30. R. C. Ward |
| 2. J. B. Drake | 31. P. H. Worley |
| 3. G. A. Geist | 32. A. Zucker |
| 4-5. R. F. Harbison | 33. J. J. Dorning (Consultant) |
| 6. M. T. Heath | 34. R. M. Haralick (Consultant) |
| 7-11. J. K. Ingersoll | 35. Central Research Library |
| 12. M. R. Leuze | 36. ORNL Patent Office |
| 13-17. F. C. Maienschein | 37. K-25 Plant Library |
| 18. E. G. Ng | 38. Y-12 Technical Library |
| 19. G. Ostrouchov | /Document Reference Station |
| 20-24. B. W. Peyton | 39. Laboratory Records - RC |
| 25. C. H. Romine | 40-41. Laboratory Records Department |

EXTERNAL DISTRIBUTION

42. Mr. C. Cleveland Ashcraft, Department of Computer Science, Yale University, P.O. Box 7158, New Haven, CT 06520
43. Dr. Donald M. Austin, Office of Scientific Computing, Office of Energy Research, ER-7, Germantown Building, U.S. Department of Energy, Washington, DC 20545
44. Lawrence J. Baker, Exxon Production Research Company, P.O. Box 2189, Houston, TX 77252-2189
45. Dr. Jesse L. Barlow, Department of Computer Science, Pennsylvania State University, University Park, PA 16802
46. Dr. Chris Bischof, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
47. Prof. Ake Bjorck, Department of Mathematics, Linkoping University, Linkoping 58183, Sweden
48. Dr. Jean R. S. Blair, Department of Computer Science, Ayres Hall, University of Tennessee, Knoxville, TN 37996-1301

49. Dr. James C. Browne, Department of Computer Sciences, University of Texas, Austin, TX 78712
50. Dr. Bill L. Buzbee, Scientific Computing Division, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307
51. Dr. Donald A. Calahan, Department of Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI 48109
52. Dr. Tony Chan, Department of Mathematics, University of California, Los Angeles, 405 Hilgard Avenue, Los Angeles, CA 90024
53. Dr. Jagdish Chandra, Army Research Office, P.O. Box 12211, Research Triangle Park, North Carolina 27709
54. Dr. Eleanor Chu, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
55. Dr. Mel Ciment, National Science Foundation, 1800 G Street, NW, Washington, DC 20550
56. Prof. Tom Coleman, Department of Computer Science, Cornell University, Ithaca, NY 14853
57. Dr. Paul Concus, Mathematics and Computing, Lawrence Berkeley Laboratory, Berkeley, CA 94720
58. Dr. Jane K. Cullum, IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598
59. Dr. George Cybenko, Computer Science Department, University of Illinois, Urbana, IL 61801
60. Dr. George J. Davis, Department of Mathematics, Georgia State University, Atlanta, GA 30303
61. Dr. Jack J. Dongarra, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
62. Dr. Iain Duff, CSS Division, Harwell Laboratory, Didcot, Oxon OX11 0RA, England
63. Prof. Pat Eberlein, Department of Computer Science, SUNY/Buffalo, Buffalo, NY 14260

64. Dr. Stanley Eisenstat, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520
65. Dr. Lars Elden, Department of Mathematics, Linkoping University, 581 83 Linkoping, Sweden
66. Dr. Howard C. Elman, Computer Science Department, University of Maryland, College Park, MD 20742
67. Mr. Robert E. England, Department of Computer Science, Ayres Hall, University of Tennessee, Knoxville, TN 37996-1301
68. Dr. Albert M. Erisman, Boeing Computer Services, MS 7L-21, P.O. Box 24346, Seattle, WA 98124-0346
69. Dr. Geoffrey C. Fox, Booth Computing Center 158-79, California Institute of Technology, Pasadena, CA 91125
70. Dr. Paul O. Frederickson, NASA Ames Research Center, RIACS, M/S 230-5, Moffett Field, CA 94035
71. Dr. Fred N. Fritsch, L-300, Mathematics and Statistics Division, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550
72. Dr. Robert E. Funderlic, Department of Computer Science, North Carolina State University, Raleigh, NC 27650
73. Dr. Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47405
74. Dr. David M. Gay, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
75. Dr. C. William Gear, Computer Science Department, University of Illinois, Urbana, Illinois 61801
76. Dr. Alan George, Vice President, Academic and Provost, Needles Hall, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
77. Dr. John Gilbert, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304
78. Prof. Gene H. Golub, Department of Computer Science, Stanford University, Stanford, CA 94305

79. Dr. Joseph F. Grcar, Division 8331, Sandia National Laboratories, Livermore, CA 94550
80. Dr. Per Christian Hansen, UCI*C Lyngby, Building 305, Technical University of Denmark, DK-2800 Lyngby, Denmark
81. Dr. Don E. Heller, Physics and Computer Science Department, Shell Development Co., P.O. Box 481, Houston, TX 77001
82. Dr. Charles J. Holland, Air Force Office of Scientific Research, Building 410, Bolling Air Force Base, Washington, DC 20332
83. Dr. Robert E. Huddleston, Computation Department, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550
84. Dr. Ilse Ipsen, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520
85. Ms. Elizabeth Jessup, Department of Computer Science, Yale University, P.O. Box 2158, Yale Station, New Haven, CT 06520
86. Dr. Harry Jordan, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO 80309
87. Dr. Bo Kagstrom, Institute of Information Processing, University of Umea, 5-901 87 Umea, Sweden
88. Dr. Hans Kaper, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
89. Dr. Linda Kaufman, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
90. Dr. Robert J. Kee, Applied Mathematics Division 8331, Sandia National Laboratories, Livermore, CA 94550
91. Dr. Alan J. Laub, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106
92. Dr. Robert L. Launer, Army Research Office, P.O. Box 12211, Research Triangle Park, North Carolina 27709
93. Dr. Charles Lawson, MS 301-490, Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109

94. Prof. Peter D. Lax, Director, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012
95. Dr. John G. Lewis, Boeing Computer Services, P.O. Box 24346, M/S 7L-21, Seattle, WA 98124-0346
96. Dr. Heather M. Liddell, Director, Center for Parallel Computing, Department of Computer Science and Statistics, Queen Mary College, University of London, Mile End Road, London E1 4NS, England
97. Dr. Joseph Liu, Department of Computer Science, York University, 4700 Keele Street, North York, Ontario, Canada M3J 1P3
98. Dr. Franklin Luk, Electrical Engineering Department, Cornell University, Ithaca, NY 14853
99. Dr. Thomas A. Manteuffel, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545
100. Dr. Paul C. Messina, California Institute of Technology, Mail Code 158-79, Pasadena, CA 91125
101. Dr. Cleve Moler, Ardent Computers, 550 Del Ray Avenue, Sunnyvale, CA 94086
102. Dr. Brent Morris, National Security Agency, Ft. George G. Meade, MD 20755
103. Dr. Dianne P. O'Leary, Computer Science Department, University of Maryland, College Park, MD 20742
104. Maj. C. E. Oliver, Office of the Chief Scientist, Air Force Weapons Laboratory, Kirtland Air Force Base, Albuquerque, NM 87115
105. Dr. James M. Ortega, Department of Applied Mathematics, University of Virginia, Charlottesville, VA 22903
106. Prof. Chris Paige, Department of Computer Science, McGill University, 805 Sherbrooke Street W., Montreal, Quebec, Canada H3A 2K6
107. Prof. Roy P. Pargas, Department of Computer Science, Clemson University, Clemson, SC 29634-1906
108. Prof. Beresford N. Parlett, Department of Mathematics, University of California, Berkeley, CA 94720

109. Prof. Merrell Patrick, Department of Computer Science, Duke University, Durham, NC 27706
110. Dr. Robert J. Plemmons, Departments of Mathematics and Computer Science, North Carolina State University, Raleigh, NC 27650
111. Dr. Alex Pothan, Department of Computer Science, Pennsylvania State University, University Park, PA 16802
112. Dr. John K. Reid, CSS Division, Building 8.9, AERE Harwell, Didcot, Oxon, England OX11 0RA
113. Dr. John R. Rice, Computer Science Department, Purdue University, West Lafayette, IN 47907
114. Dr. Garry Rodrigue, Numerical Mathematics Group, Lawrence Livermore Laboratory, Livermore, CA 94550
115. Dr. Donald J. Rose, Department of Computer Science, Duke University, Durham, NC 27706
116. Dr. Ahmed H. Sameh, Computer Science Department, University of Illinois, Urbana, IL 61801
117. Dr. Michael Saunders, Systems Optimization Laboratory, Operations Research Department, Stanford University, Stanford, CA 94305
118. Dr. Robert Schreiber, RIACS, Mail Stop 230-5, NASA Ames Research Center, Moffet Field, CA 94035
119. Dr. Martin H. Schultz, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520
120. Dr. David S. Scott, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006
121. Dr. Lawrence F. Shampine, Mathematics Department, Southern Methodist University, Dallas, TX 75275
122. Dr. Kermit Sigmon, Department of Mathematics, University of Florida, Gainesville, FL 32611
123. Dr. Horst Simon, Mail Stop 258-5, NASA Ames Research Center, Moffett Field, CA 94035

124. Dr. Danny C. Sorensen, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
125. Prof. G. W. Stewart, Computer Science Department, University of Maryland, College Park, MD 20742
126. Dr. Michael G. Thomason, Department of Computer Science, Ayres Hall, University of Tennessee, Knoxville, TN 37996-1301
127. Prof. Charles Van Loan, Department of Computer Science, Cornell University, Ithaca, NY 14853
128. Dr. Robert G. Voigt, ICASE, MS 132-C, NASA Langley Research Center, Hampton, VA 23665
129. Dr. Daniel D. Warner, Department of Mathematical Sciences, O-104 Martin Hall, Clemson University, Clemson, SC 29631
130. Dr. Andrew B. White, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545
131. Dr. Arthur Wouk, Army Research Office, P.O. Box 12211, Research Triangle Park, North Carolina 27709
132. Dr. Margaret Wright, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
133. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P.O. Box 2001 Oak Ridge, TN 37831-8600
- 134-143. Office of Scientific & Technical Information, P.O. Box 62, Oak Ridge, TN 37831