COMPACT CLIQUE TREE DATA STRUCTURES IN SPARE MATRIX FACTORIZATIONS

by

Alex Pothen
and
Chungguang Sun

Computer Sciences Technical Report #897

December 1989

COMPACT CLIQUE TREE DATA STRUCTURES IN SPARSE MATRIX FACTORIZATIONS [*]

ALEX POTHEN[†] AND CHUNGUANG SUN[‡]

Abstract. The design of compact data structures for representing the structure of the Cholesky factor L of a sparse, symmetric positive definite matrix A is considered. The clique tree data structure described in [10] provides a compact representation when the structure of L must be stored explicitly. Two more compact data structures, the compact clique tree and the skeleton clique tree, which represent the structure of L implicitly, i.e., when some computation on the data structure is permitted to obtain the structure of L, are described. The compact clique tree is computed from a clique tree of L, while the skeleton clique tree is computed from the skeleton matrix introduced by Liu [12] and a subset of the information in the clique tree. The relationships between these data structures are considered, and it is proved that the compact clique tree is the smaller of the two. Computational results on some Boeing-Harwell test problems show that these data structures require less space than either the clique tree or the matrix A. It is also more efficient to generate the structure of L from these data structures than by symbolic factorization on A. The final Section contains a brief discussion of applications, related work, and some questions raised by this work.

AMS(MOS) subject classifications: 65F50, 65F25, 68R10.

Keywords. chordal graph, clique tree, compact clique tree, sparse Cholesky factorization, sparse matrix, skeleton matrix, skeleton clique tree.

[*] Presented at the Workshop on Large-Scale Numerical Optimization, Mathematical Sciences Institute, Cornell University, Oct 18–20, 1989.

[†] Departments of Computer Science and Mathematics, The University of Wisconsin-Madison, 1210 W. Dayton St., Madison, WI 53706. Electronic address: pothen@cs.wisc.edu, na.pothen@na-net.stanford.edu. Permanent address: Department of Computer Science, Whitmore Lab, The Pennsylvania State University, University Park, PA 16802. The research of this author was supported by NSF grant CCR-8701723, NSF equipment grant CCR-8705110, and U.S. Air Force Office of Scientific Research grant AFOSR-88-0161.

[‡] Department of Computer Science, Whitmore Lab, The Pennsylvania State University, University Park, PA 16802. Electronic address: csun@shire.cs.psu.edu. The research of this author was supported by U.S. Air Force Office of Scientific Research grant AFOSR-88-0161.

Table Of Contents

COMPACT CLIQUE TREE DATA STRUCTURES IN SPARSE MATRIX FACTORIZATIONS [*]

ALEX POTHEN[†] AND CHUNGUANG SUN[‡]

Abstract. The design of compact data structures for representing the structure of the Cholesky factor L of a sparse, symmetric positive definite matrix A is considered. The clique tree data structure described in [10] provides a compact representation when the structure of L must be stored explicitly. Two more compact data structures, the compact clique tree and the skeleton clique tree, which represent the structure of L implicitly, i.e., when some computation on the data structure is permitted to obtain the structure of L, are described. The compact clique tree is computed from a clique tree of L, while the skeleton clique tree is computed from the skeleton matrix introduced by Liu [12] and a subset of the information in the clique tree. The relationships between these data structures are considered, and it is proved that the compact clique tree is the smaller of the two. Computational results on some Boeing-Harwell test problems show that these data structures require less space than either the clique tree or the matrix A. It is also more efficient to generate the structure of L from these data structures than by symbolic factorization on A. The final Section contains a brief discussion of applications, related work, and some questions raised by this work.

AMS(MOS) subject classifications: 65F50, 65F25, 68R10.

Keywords. chordal graph, clique tree, compact clique tree, sparse Cholesky factorization, sparse matrix, skeleton matrix, skeleton clique tree.

1. Introduction.

Consider the Cholesky factorization of a large, sparse, symmetric positive definite matrix A. What is the least space in which the nonzero structure of the Cholesky factor L can be stored? The answer to this question has implications in the design of compact data structures for several sparse matrix problems. There are two versions in which this question can be posed.

In the first, we are required to compute a data structure from which the structure of L can be obtained without any additional computation. In this case, it is possible to store the structure of L in space much smaller than the number of nonzeros in L by using the clique tree data structure described in [10]. The existence of this data structure is the primary reason for the spectacular success of the *compressed column storage scheme* proposed by Sherman [16] and used in modern sparse matrix software. The size of the clique tree data structure is the number of compressed row subscripts in a scheme which stores L by columns, when 'accidental compression' is ignored.

In the second version of the question, we are required to compute a compact data structure for L such that some 'reasonable' amount of computation on the data structure is permitted to obtain the structure of L. This question is relevant in contexts in which the

[*] Presented at the Workshop on Large-Scale Numerical Optimization, Mathematical Sciences Institute, Cornell University, Oct 18–20, 1989.

[†] Departments of Computer Science and Mathematics, The University of Wisconsin-Madison, 1210 W. Dayton St., Madison, WI 53706. Electronic address: pothen@cs.wisc.edu, na.pothen@na-net.stanford.edu. Permanent address: Department of Computer Science, Whitmore Lab, The Pennsylvania State University, University Park, PA 16802. The research of this author was supported by NSF grant CCR-8701723, NSF equipment grant CCR-8705110, and U.S. Air Force Office of Scientific Research grant AFOSR-88-0161.

[‡] Department of Computer Science, Whitmore Lab, The Pennsylvania State University, University Park, PA 16802. Electronic address: csun@shire.cs.psu.edu. The research of this author was supported by U.S. Air Force Office of Scientific Research grant AFOSR-88-0161.

structure of L must be generated, used, and then discarded (to keep storage requirements low) several times in the course of a computation. In this case, one possible choice for a data structure is the structure of A; a symbolic factorization step can then be performed every time the structure of L is required. In this paper, we describe two data structures, the *compact clique tree* and the *skeleton clique tree*, which provide more compact representations of the structure of L. We prove that these data structures require space bounded by the smaller of the number of nonzeros in A and the number of compressed subscripts in L; and in practice, the space required for these data structures is much less than the bound. It will also turn out that it is faster to generate the structure of L from these data structures than from A.

Both the compact clique tree and the skeleton clique tree are related to the clique tree data structure. The former is computed directly from the clique tree using a certain property of clique trees. The skeleton clique tree is also related to the concept of the skeleton matrix introduced by Liu [12]. This data structure is obtained by computing a small subset of the information in the clique tree together with the information in the skeleton matrix. We will prove that the compact clique tree is the smaller of the two data structures, though in practice the two require almost the same amount of storage.

We do not know if either the clique tree or the compact clique tree are optimal data structures with respect to space for representing L structurally (the former when no computation is permitted on the data structure, and the latter when some computation is permitted). Barring 'accidental compression' (observed, for instance, in a symbolic factorization procedure), it is likely that the clique tree is an optimal data structure which provides an answer for the first question.

The compact clique tree and skeleton clique tree data structures provide compact column-oriented storage schemes for the Cholesky factor L. In this sense, they complement the compact row-oriented storage scheme for L designed by Liu [12] which makes use of the skeleton matrix.

An outline of this paper is as follows. § 2 contains a description of the clique tree data structure as defined in [10]. Since a definition of a clique tree requires several concepts about the maximal clique structure of chordal graphs, the relevant notions are also briefly reviewed. In § 3 we design a new algorithm for computing a clique tree. This algorithm computes a subset of the information in the clique tree from the elimination tree without requiring a symbolic factorization step. This algorithm will be used later with some modifications to compute a skeleton clique tree. We explore the relations between clique trees and elimination trees in § 4. A major result of this section is that there exists a path between two vertices in the elimination tree if and only if there is a corresponding path in a clique tree. The results from this section will be necessary later to study the relationships between compact clique trees and skeleton clique trees. § 5 contains the descriptions of the compact clique tree and the skeleton clique tree data structures. We include in this section a brief description of the skeleton matrix, and show how the new clique tree algorithm can be modified to compute a skeleton clique tree. The relationships between these two data structures are investigated in § 6. We characterize the situations where the compact cliques (the vertices of the compact clique tree) and the skeleton cliques (the vertices of the skeleton clique tree) differ from each other. Computational results on the sizes of these data structures and the times required to compute them for several large problems from the Boeing-Harwell collection are reported in § 7. The final § 7 briefly discusses some applications, related work, and some questions raised by this research.

We employ the following notation. The order of the matrix A is n, which is also the number of vertices in the adjacency graph $G(A)$. Without loss of generality, we assume that $G(A)$ is connected. We denote the adjacency graph of the Cholesky factor L (the filled graph) by G. The number of nonzeros in the strict lower triangle of A (L) will be denoted by $|A|$ ($|L|$). Let K_1, K_2, …, K_m denote the maximal cliques of the filled graph G. The number of maximal cliques is m, and q denotes the size of a clique tree; i.e., this is the sum $\sum_{i=1}^{n} |K_i|$. Let the vertices of a graph be numbered in some ordering from 1 to n. We denote the set of vertices adjacent to a vertex v and numbered higher than v (the higher adjacency set) by $hadj(v)$. Similarly, the set $ladj(v)$ will denote the set of vertices adjacent to v and numbered lower than v (the lower adjacency set). The number $hd(v)$ (higher degree) will denote the cardinality of the set $hadj(v)$.

2. Background on clique trees.

In this section, we review the concept of a clique tree as described in [10], and describe an algorithm which was designed there for computing a clique tree from the output of a symbolic factorization routine. Throughout this section, we assume the following context. A sparse symmetric matrix has been ordered by a fill-reducing ordering algorithm, and the rows and columns of the matrix have been reordered in this fill-reducing ordering. The adjacency graph G of the Cholesky factor L is chordal, and as a result of the reordering, the natural ordering of the vertices of G is a perfect elimination ordering (peo). In addition, we assume that an efficient symbolic factorization procedure has been used to compute the nonzero structures of the columns of the factor matrix L; thus the higher adjacency lists of the vertices $\{hadj(v) : v \in V\}$ are available.

Maximal cliques. The vertices of a clique tree are the maximal cliques of the chordal graph G, and therefore, the first step in computing a clique tree is to identify the maximal cliques of the filled graph. They are easily identified by the following characterization due to Fulkerson and Gross [6].

PROPOSITION 2.1. *Let K be a maximal clique in a chordal graph G whose vertices are numbered in a peo. If v is the lowest-numbered vertex in K, then $K = v + hadj(v)$.*

The maximal cliques of G can hence be represented as $K(v_1)$, $K(v_2)$, …, $K(v_m)$, where $K(v)$ specifies the maximal clique $v + hadj(v)$ generated by the lowest-numbered or *representative* vertex v. If a vertex is not the representative of *any* maximal clique, we call it a *non-representative* vertex. From Proposition 2.1, it is only necessary to identify all representative vertices to obtain the maximal cliques of G.

A chordal graph which arises from a finite element formulation is shown in Figure 1. The reader can verify that its maximal cliques are as follows:

$$K(1) = \{1, 2, 5, 10\}, \quad K(3) = \{3, 4, 5, 6\},$$
$$K(5) = \{5, 6, 10, 11\}, \quad K(7) = \{7, 8, 9, 10, 11\}.$$

We now turn our attention to a method for identifying representative vertices.

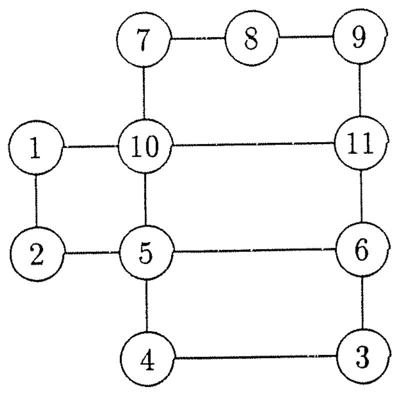

FIG. 1. *A chordal graph from a finite element problem. Each internal face of this planar graph is a clique; such 'diagonal' edges have not been drawn for clarity.*

PROPOSITION 2.2 ([10]). *A vertex v is not a representative vertex for any maximal clique if and only if there exists a representative vertex z numbered lower than v, with $v + hadj(v) \subset z + hadj(z)$.*

The following Proposition provides an efficient test of whether the set $v + hadj(v)$ is contained in a set $z + hadj(z)$, using only the higher degrees of the vertices.

PROPOSITION 2.3 ([10]). *Let $K(v) = \{v \equiv w_0 < w_1 < \ldots < w_p\}$ be a maximal clique and let w_i be the highest-numbered vertex in $K(v)$ such that $hd(w_j) = p - j$ for $0 \le j \le i$. Then $\{w_1, \ldots, w_i\}$ are not representative vertices.*

Note that since the vertices are numbered in peo, when w_0 is eliminated, all of its higher-numbered neighbors are adjacent to w_1. Hence $hadj(w_0) \subseteq w_1 + hadj(w_1)$. By repeating this argument for w_1, ..., w_{p-1}, we have

$$hadj(w_0) \subseteq w_1 + hadj(w_1) \subseteq \{w_1, w_2\} \cup hadj(w_2) \ldots \subseteq \{w_1, \ldots, w_p\} \cup hadj(w_p).$$

Hence the higher degrees satisfy $hd(w_0) \le 1 + hd(w_1) \le \ldots \le p + hd(w_p)$. If $p = hd(w_0) = i + hd(w_i)$ for some w_i with $1 \le i \le p$, it follows that

$$hadj(w_0) = w_1 + hadj(w_1) = \ldots = \{w_1, \ldots, w_i\} \cup hadj(w_i).$$

Thus $w_j + hadj(w_j) \subset w_0 + hadj(w_0)$, for $1 \le j \le i$. Further, by the choice of i, if $(i+1) \le k \le p$, then $p = hd(w_0) < k + hadj(w_k)$, and $w_k + hadj(w_k) \not\subset w_0 + hadj(w_0)$.

These results can be used to design an algorithm for computing the representative vertices of the maximal cliques of G. Initially we consider all nodes are possible representatives, and then process the vertices in the given peo. To process a vertex v, we identify if it is a representative vertex, and if so, mark vertices which can be identified as nonrepresentative vertices in $hadj(v)$. In more detail: If a vertex v has not been marked as non-representative by the time it is processed, no earlier vertex can satisfy the role of z in Proposition 2.2 and so we accept v as a representative vertex. We then examine $hadj(v)$ and mark vertices satisfying the degree test in Proposition 2.3 as non-representative vertices, stopping when a vertex fails the degree test.

The above method marks a vertex as a nonrepresentative vertex in each maximal clique in which it can do so. A more efficient method marks each nonrepresentative vertex w

exactly once, in the first maximal clique in which w can be identified as a nonrepresentative vertex. This can be accomplished by stopping the search for nonrepresentative vertices in $hadj(v)$ as soon as we find either a vertex that fails the degree test or a vertex that has already been marked as a nonrepresentative. It can be proved (Proposition 5 in [10]) that the more efficient method will not fail to mark any other vertices which are identifiable as nonrepresentative in $hadj(v)$.

A partition of maximal cliques. The more efficient method for identifying representative vertices can also be used to partition the vertices in a maximal clique into two sets which will be useful for computing a clique tree. Let v be the representative vertex of a maximal clique $K = K(v)$. We partition the vertices in K into two sets: $new(K)$ and $anc(K)$. (The notation adopted for these sets will become clear when a clique tree is constructed from them.) The set $new(K)$ (read 'new set of K') consists of the representative vertex v together with the vertices marked as non-representative while examining $hadj(v)$. The set $anc(K)$ (read 'ancestor set of K') consists of the vertices in K that either fail the degree test or were marked as non-representative in some other maximal clique $K(u)$, with $u < v$. We also define a vertex $first\_anc(K)$ (read 'first-ancestor vertex in K'), which is the last vertex examined in $hadj(v)$, if $anc(K)$ is not the emptyset. If all vertices in $hadj(v)$ are marked as nonrepresentative, then $anc(K)$ is empty, and $first\_anc(K)$ is undefined.

The reader can verify that the maximal cliques of the chordal graph in Fig. 1 are partitioned as follows:

$$\begin{array}{ll} new(K(1)) = \{1, 2\}, & anc(K(1)) = \{5, 10\}; \\ new(K(3)) = \{3, 4\}, & anc(K(3)) = \{5, 6\}; \\ new(K(5)) = \{5, 6, 10, 11\}, & anc(K(5)) = \emptyset; \\ new(K(7)) = \{7, 8, 9\}, & anc(K(7)) = \{10, 11\}. \end{array}$$

The partition of a maximal clique into the sets $new(K)$ and $anc(K)$ has two important properties that we will use later in defining a clique tree from the maximal cliques. First, note that the vertices in $hadj(v)$ are examined in peo, and the marking process stops when a vertex fails the degree test or when a previously marked vertex is encountered. Hence 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 vertex. Second, the sets $new(K(v_1))$, $new(K(v_2))$, …, $new(K(v_m))$ form a partition of the nodes of the chordal graph, since every node is chosen as a representative vertex or marked as a nonrepresentative vertex exactly once.

A clique tree. We now have all the information necessary to define a clique tree. The vertices of the clique tree are the maximal cliques of the chordal graph G. Let w_l be the $first\_anc$ vertex for a maximal clique $K(v)$ with $anc(K(v)) \neq \emptyset$. Then the parent clique of $K(v)$ is the clique P that contains w_l as a *new* vertex. Such a clique P exists, since every vertex belongs as a *new* vertex in some maximal clique, by the partitioning property of the *new* sets. The root of the clique tree is the clique R which satisfies $new(R) = R$, and $anc(R) = \emptyset$; $first\_anc(R)$ is undefined. The root clique R necessarily contains the vertex numbered n as a *new* vertex.

Fig. 2 shows the clique tree of the chordal graph in Fig. 1 computed from the *new* and *anc* sets obtained previously. The partitioning of a maximal clique is shown by drawing the

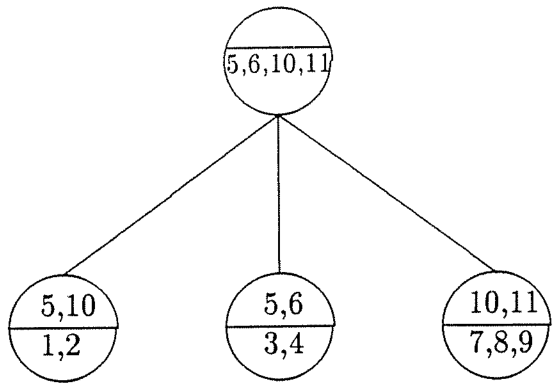

FIG. 2. *A clique tree of the chordal graph in Figure 1 computed by Clique Tree Algorithm* 1.

new sets at the bottom and the *anc* sets at the top. The reader can verify that the *first_anc* vertex of each maximal clique is a *new* vertex in its parent clique, and that the root clique has its *anc* set empty.

An algorithm which implements the above method was designed in [10]; we shall call this the Clique Tree Algorithm 1. It computes the representative vertices, the *first_anc* vertices of the maximal cliques, and a clique tree from the higher adjacency lists and the higher degrees of the vertices of a chordal graph. If v is the representative of a maximal clique $K(v)$, a pointer to the *first_anc* vertex in the $hadj(v)$ list is all that is necessary to obtain the $new(K(v))$, $anc(K(v))$ partition, since these are ordered subsets of $v + hadj(v)$. The algorithm also computes a mapping $new\_in\_clique(v)$ for each vertex v, which identifies the unique clique in which v is a *new* vertex. This algorithm requires $O(n)$ time and $O(q)$ space to compute a clique tree.

Properties of clique trees. We now consider the properties of clique trees which will be useful in the study of compact clique tree data structures. At the outset, we note that the above definition of a clique tree is a more restrictive definition than has been previously employed in other contexts. Let P denote the parent clique of a maximal clique K. As a consequence of our definition of a clique tree, it can be proved (Proposition 6 and Theorem 7, [10]) that

- $anc(K) \subset P$, $new(K) \cap P = \emptyset$;
- a vertex $w \in new(K)$ can belong to the *anc* set of another clique D only if D is a descendant of K; and
- a vertex $w \in anc(K)$ belongs to the *new* set of some ancestor clique A; further, w belongs to all cliques which lie on the path from K to A in the clique tree.

These properties characterize 'legal' clique trees of chordal graphs. With our more restrictive definition, we acquire the additional property that the parent P is the *only* clique which contains all the vertices in $anc(K)$—all other ancestor cliques can contain only proper subsets of $anc(K)$.

It is instructive to consider the significance of the partition of each maximal clique K into the $new(K)$ and $anc(K)$ sets. Let $\{v_1 < v_2 < \ldots < v_m\}$ be the set of representative vertices of the maximal cliques of G. The Clique Tree Algorithm 1 then implicitly orders the maximal cliques by their representative vertices: $K(v_1) < K(v_2) < \ldots < K(v_m)$. This ordering of the maximal cliques partitions each maximal clique $K(v)$ as follows.

$$\begin{aligned} new(K(v)) &= \{w \in K(v) : hadj(w) \subset K(v), \text{ but } hadj(w) \not\subset K(u), \\ &\quad \text{where } K(u) < K(v)\}. \\ anc(K(v)) &= K(v) \setminus new(K(v)). \end{aligned}$$

The reason for calling these sets *new* and *anc* sets can be explained with respect to the clique tree. We can consider the chordal graph as being 'built up' by adding one maximal clique at a time, starting with the root clique in the clique tree, and proceeding down the clique tree level by level. When a maximal clique K is added, $anc(K)$ is the set of vertices that also belong to some ancestor cliques of K, and the vertices in $new(K)$ have not appeared in any of the cliques up to this level. These latter vertices may appear in some descendant cliques of K.

We should mention that that the Clique Tree Algorithm 1 is only one of many algorithms which may be designed to compute clique trees. The other algorithms differ from the Clique Tree Algorithm 1 in the ordering of the maximal cliques. It should be clear from the definitions of the *new* and *anc* sets that if the ordering of the maximal cliques is changed, these sets and the clique tree may change too.

3. A new clique tree algorithm.

In Section 5 of this paper, we design an algorithm to compute a compact data structure for the structure of L called the skeleton clique tree. In this context (and several others), we wish to compute only a subset of the information in a clique tree. More precisely, we require only the representative vertices of the maximal cliques and the parent relationships among the maximal cliques. Since we define a parent clique in terms of the *new_in_clique* mapping and the *first_anc* vertices, we need compute only the representative and the *first_anc* vertices, and the *new_in_clique* mapping. In particular, the *anc* sets are not required, and hence this subset of the information in a clique tree can be represented in $O(n)$ space.

A natural question is the following: Is it possible to compute a clique tree (i.e., the information listed in the previous paragraph) without performing symbolic factorization to obtain the higher adjacency lists of the vertices? In this section, we design an algorithm to compute a clique tree from the elimination tree and the higher degrees of the vertices in $O(|A|)$ space and $O(n)$ time. (The elimination tree and the higher degrees can be computed directly from A without symbolic factorization.) Throughout this section, we will refer to an algorithm for computing the above subset of information about a clique tree as a clique tree algorithm.

Elimination trees. The elimination tree $F = (V, E)$ (henceforth *etree*) has its vertex set V equal to the vertices of the graph G. For each $v \in V$ such that $hadj(v) \neq \emptyset$, E contains an edge (v, w) where w is the lowest-numbered vertex in $hadj(v)$. The vertex w is the *parent* of v in the etree. The root of the etree is the vertex numbered n, since its higher adjacency set is empty. Since each child is numbered lower than the parent, this is a heap-ordered tree. An etree of the graph in Fig. 1 is given in Fig. 3.

The etree captures the column dependencies in the Cholesky factorization of the matrix A. Hence it plays an important role in modern sparse matrix algorithms, and Liu [13]

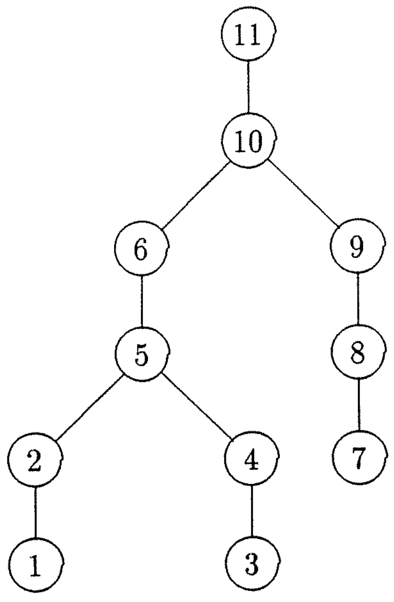

FIG. 3. *The elimination tree of the graph in Fig.1*

provides an excellent survey of this role. Liu [12] and Zmijewski and Gilbert [17] have designed efficient algorithms for computing the etree directly from A; the asymptotically fastest algorithm runs in $O(|A|\alpha(|A|, n))$ time, where $\alpha(*, *)$ is the functional inverse of Ackermann's function. The higher degrees of the vertices can then be computed from A and the etree in $O(|L|)$ time.

A new clique tree algorithm. Now we describe the Clique Tree Algorithm 2, which computes a clique tree from the etree and the higher degrees of the vertices. The algorithm processes the vertices of the etree from 1 to n. If a vertex v has a child in the etree with $1 + hd(v) = hd(u)$, then $v + hadj(v) \subset u + hadj(u)$, v is not a representative vertex, and v is made a *new* vertex in the clique K_u which includes u as a *new* vertex. If there are several such children, we choose one child u arbitrarily. If v is an interior vertex of the etree with no such child, or if v is a leaf of the etree, we make v the representative vertex of a new maximal clique K_i. Finally, if v is the representative vertex of K_i, for each child s of v in the etree included in K_s as a *new* vertex, we make K_s a child of K_i in the clique tree. The Clique Tree Algorithm 2 is shown in Fig. 4.

It is clear that the algorithm requires $O(n)$ time, since each vertex v in the etree is examined at most twice: once while processing v, and once while processing the parent of v in the etree.

When a vertex v has several children in the etree satisfying the degree test, v can be included as a *new* vertex in any one of the maximal cliques in which a child is included as a *new* vertex. This freedom can be used by the Clique Tree Algorithm 2 to compute clique trees with differing properties.

A clique tree of the filled graph in Fig. 1 (with corresponding etree in Fig. 3) computed by the Clique Tree Algorithm 2 is shown in Fig. 5. This algorithm can also compute the clique tree shown in Fig. 2. The two different clique trees are obtained by choosing to merge vertex 10 into either the clique which includes its child 9 as a *new* vertex, or the clique which includes its other child 6 as a *new* vertex.

```
$i := 0$;
for $v := 1$ to $n \rightarrow$
    if $v$ has a child $u$ in etree with $hd(v) + 1 = hd(u)$ then
        $K_u := new\_in\_clique(u)$;
        $new\_in\_clique(v) := K_u$; $new(K_u) := new(K_u) + v$;
    else
        $i := i + 1$;
        make $v$ the representative of a maximal clique $K_i = v + hadj(v)$;
        $new\_in\_clique(v) := K_i$; $new(K_i) := \{v\}$;
    fi
    for each child $s$ of $v$ in etree with $new\_in\_clique(s) \neq new\_in\_clique(v) \rightarrow$
        $K_s := new\_in\_clique(s)$; $first\_anc(K_s) := v$;
        make $K_s$ a child of $new\_in\_clique(v)$ in the clique tree;
    rof
rof
```

FIG. 4. *The Clique Tree Algorithm 2. This algorithm computes the representative and first-ancestor vertices, and the sets of new vertices of the maximal cliques from the etree and the higher degrees.*

We omit a proof of correctness of the Clique Tree Algorithm 2 for brevity. Instead, we discuss the differences between the two clique tree algorithms, since the second algorithm can be proved correct by a proof similar to the proof of correctness of the Clique Tree Algorithm 1, provided in [10].

Recall that the Clique Tree Algorithm 1 implicitly orders the maximal cliques by their representative vertices. It then partitions each clique K into two sets $new(K)$ and $anc(K)$ as described in § 2. The major difference in Clique Tree Algorithm 2 is that the above ordering of the maximal cliques is abandoned. For each maximal clique $K = \{w_0 < w_1 < \ldots < w_p\}$ (except the root clique), the Clique Tree Algorithm 2 identifies two sets $new(K)$, $anc(K)$ and a vertex $first\_anc(K)$ such that

1. $new(K) = \{w_0 < w_1 < \ldots < w_{l-1}\}$, where $1 \leq l \leq p$; and

$$hadj(w_0) = w_1 + hadj(w_1) = \ldots = \{w_0, w_1, \ldots, w_{l-1}\} \cup hadj(w_{l-1}).$$

2. $first\_anc(K) = w_l$ satisfies either (i). $hadj(w_l) \not\subset K$, or (ii). $hadj(w_l) \subset K$, but there exists exactly one other maximal clique K' satisfying $w_l \in new(K')$, and $hadj(w_l) \subset K'$; and
3. $anc(K) = \{w_l < w_{l+1} < \ldots < w_p\}$.

For the root clique R, $new(R) = R$, $anc(R) = \emptyset$, and $first\_anc(R)$ is undefined.

The parent relationships are defined exactly as in the Clique Tree Algorithm 1: P is the parent of K if $first\_anc(K)$ is *new* in P. We can then prove that $new(K) \cap P = \emptyset$ and $anc(K) \subset P$. Theorem 7 in [10] (see the discussion at the end of Section 2) shows that a tree joining the maximal cliques satisfying the above three properties is a legal clique tree. Since the same definitions of the parent relationships are used in both clique tree algorithms, the

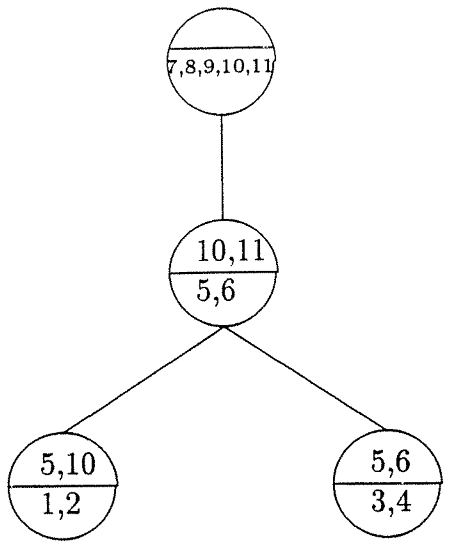

FIG. 5. *A clique tree of the chordal graph in Fig. 1 computed by Clique Tree Algorithm 2. Note that this clique tree differs from the one in Fig. 2.*

clique trees constructed by the Clique Tree Algorithm 2 also possess the other properties of clique trees discussed at the end of § 2.

Finally, we mention that the Clique Tree Algorithm 2 is similar in many respects to an algorithm for computing supernodes designed in [9] and [14].

4. Clique trees and elimination trees.

In this section, we study the relationships between paths in etrees and clique trees. We make no assumption about how the clique tree is computed—the only requirement is that the *parent* of a clique K should be defined as the clique which contains *first_anc*(K) as a *new* vertex. Our results in this section will be necessary for studying the relationships between the compact clique tree and the skeleton clique tree data structures.

The only paths we consider are paths from some vertex of either the etree or the clique tree to the root, or subpaths of such a path. (Hence these paths go 'up' in these trees, if the trees are drawn with the root at the top.) We shall refer to paths in the clique tree as *clique paths.*

Let G be a chordal graph whose vertices are ordered in some peo; let F be the etree corresponding to this ordering. The maximal cliques of G are uniquely determined, and they are independent of the peo. But the partition of a maximal clique K into the sets $new(K)$, $anc(K)$ is not necessarily unique, and hence there are several clique trees corresponding to this ordering. Let T be any clique tree which corresponds to this peo.

LEMMA 4.1. *If a maximal clique K is partitioned into the sets $new(K)$ and $anc(K)$ in a clique tree T, then the vertices in $new(K)$ form a path in the etree.*

Proof. Let the vertices in $new(K)$ be $\{v \equiv w_0 < w_1 < \ldots < w_i\}$. Then, from the discussion following Proposition 2.3,

$$hadj(w_0) = w_1 + hadj(w_1) \cdots = \{w_1, \ldots, w_i\} \cup hadj(w_i).$$

The *parent* of a vertex v in the etree is the lowest-numbered vertex in $hadj(v)$. It then follows that $parent(w_j) = w_{j+1}$, for $j = 0, \ldots, i-1$. ■

LEMMA 4.2. *Let P be the parent clique and f the first-ancestor vertex of a clique K in a clique tree T. Define*

$$\overline{new}(P) = \{w : w \in new(P), \text{ and } w \geq f\}.$$

The vertices in $new(K)$ followed by the vertices in $\overline{new}(P)$ form a path in the etree.

Proof. Let e denote the highest-numbered vertex in $new(K)$. Then, again from the discussion following Proposition 2.3, $hadj(e) = anc(K)$. Since f is the first-ancestor vertex of K, f is the lowest-numbered vertex in $anc(K)$. Hence f is the parent of e in the etree. From Lemma 4.1, vertices in $new(K)$ form a path in the etree, as do vertices in $\overline{new}(P)$. The lowest-numbered vertex in the latter path is f, since f is a *new* vertex in P, by the definition of a clique tree. The former path concatenated with the edge (e, f) and the path $\overline{new}(P)$ constitutes the desired path. ■

THEOREM 4.3. *Let G be a chordal graph whose vertices are ordered in some peo; let F be the etree corresponding to this ordering; and let T be any clique tree corresponding to this peo. Then there is a path from a vertex v to a vertex $w > v$ in the etree F if and only if either (1). $v, w \in new(K)$ for some maximal clique K in T, or (2). $v \in new(C_i)$, $w \in new(K)$, and there exists a path $C_i, C_{i-1}, \ldots, C_1, K$ in T with $w \geq f_1 \equiv first\_anc(C_1)$.*

Proof. First we prove the if part. If (1) holds, by Lemma 4.1 the result is true. Hence assume that (2) holds. We prove the result by induction on the length of the clique path.

If the length of the clique path is one, the path is C_1, K and by Lemma 4.2, the result is true. Assume inductively that the result holds for all clique paths of length at most $i-1$. Consider a clique path of length i, with cliques listed as in the statement of the theorem, where $v \in new(C_i)$, and $w \in new(K)$. Denote $f_j = first\_anc(C_j)$, for $j = 1, \ldots, i$.

Then $f_i \in new(C_{i-1})$ by definition of the clique tree, and by the inductive hypothesis, there exists a path from f_i to w in the etree. Since $v \in new(C_i)$, and f_i is the first-ancestor vertex of C_i, by Lemma 4.2, there exists a path from v to f_i in the etree. The concatenation of the two paths gives the desired path from v to w.

Now we prove the only if part. Proceed up the etree from the vertex v to w. Since the *new* sets of the maximal cliques partition the vertices, each vertex on this path belongs to the *new* set of some maximal clique in the clique tree T. Let $C_i, C_{i-1}, \ldots, C_1, K$ be the cliques which contain the vertices on the $v - w$ path as *new* vertices, with $w \in new(K)$, and $v \in new(C_i)$. We will prove that these cliques form a path in the clique tree T.

We prove the result by induction on $(i+1)$, the number of cliques which include vertices on the $v - w$ etree path as *new* vertices. When $i = 0$, then both v and w are in $new(K)$, and condition (1) is satisfied. Assume inductively that the result is true when there are no more than i cliques on the path. Let z be the lowest-numbered vertex on the $v - w$ etree path such that $z \in new(C_{i-1})$. By the inductive hypothesis, there exists a clique path $C_{i-1}, \ldots, C_1, K$ in T with $w \in new(K)$, $z \in new(C_{i-1})$. Let a be the child of z which belongs to the $v - w$ etree path. By the choice of z, $a \in new(C_i)$; from Lemma 4.1, a is the highest-numbered vertex in $new(C_i)$. Now from the discussion immediately following Proposition 2.3, $hadj(a) = anc(C_i)$. Further, since z is the parent of a in the etree, z is the lowest-numbered vertex in $anc(C_i)$. Thus z is the first-ancestor vertex of the clique C_i. Then from the definition of the clique tree, the parent clique of C_i is C_{i-1}. Thus we have $w \in new(K)$, $v \in new(C_i)$, and the clique path C_i, C_{i-1}, …, K in the clique tree T.

It remains to prove that $w \geq f_1 \equiv \mathit{first\_anc}(C_1)$. Let f be the lowest-numbered vertex on the $v - w$ path which belongs to $new(K)$. Then $w \geq f$, and we shall prove that $f = f_1$. Let y be the child of f in the etree which lies on the $w - v$ path. By the choice of f, y is the highest-numbered vertex in $new(C_1)$. Thus $hadj(y) = anc(C_1)$, and f is the first-ancestor vertex of C_1. Hence $f = f_1$. This completes the proof. ∎

COROLLARY 4.4. *Let C_j be a clique on the clique path from C_i to $K \equiv C_0$ in Theorem 4.3, where $0 \leq j \leq i-1$, and denote $f_j \equiv \mathit{first\_anc}(C_j)$. Define*

$$\overline{new}(C_j) = \{x \in new(C_j) : x \geq f_{j+1}\}.$$

Then the etree path from v to w is a subpath of the concatenation of the etree paths $new(C_i)$, $\overline{new}(C_{i-1})$, …, $\overline{new}(C_1)$, $\overline{new}(K)$.

The proof follows from a careful reading of the proof of the Theorem, and is omitted.

To illustrate these results, consider the path from vertex 1 to vertex 10 in the etree in Fig. 3. The corresponding clique path in the clique tree in Fig. 2 is $K(1)$, $K(5)$; further, this etree path is a subpath of $new(K(1)), \overline{new}(K(5))$. Also consider the etree path from vertex 1 to 11 in Fig. 3. With respect to the clique tree in Fig. 5, this corresponds to the clique path $K(1)$, $K(5)$, $K(7)$. Also, the etree path is the path $new(K(1))$, $\overline{new}(K(5))$, $\overline{new}(K(7))$.

We now show how to obtain the structure of the v-th row of L from the clique tree.

THEOREM 4.5. *Let K be the maximal clique that contains v as a new vertex in a clique tree T, and let $\mathcal{D}$ be the set of proper descendants of K which contain v. Then*

$$ladj(v) = \cup_{D \in \mathcal{D}} new(D) \cup \{w : w \in new(K) \text{ and } w < v\}.$$

Proof. From Theorem 7 in [10] (see the discussion in the last paragraph of Section 2), since $v \in new(K)$, v can belong to another clique D only if D is a descendant of K. Also from the same Theorem, if v belongs to such a descendant clique D, then it also belongs to all cliques on the clique path from D to K. Hence there is a set of 'lowest descendants', $\mathcal{L} = \{L_1, \ldots, L_i\}$, such that for each clique L_j where $1 \leq j \leq i$, the following properties hold: (i). v belongs to L_j and to all cliques on the clique path from L_j to K; (ii). v does not belong to any descendant of L_j.

Let $\mathcal{D} = \{D_1, D_2, \ldots, D_k\}$ be the set of proper descendants of K that are reached by clique paths from each clique in $\mathcal{L}$ to K. These cliques together with K are the only cliques that contain v. Each clique D_j contains v as an anc vertex, and hence vertices in $new(D_j)$ belong to $ladj(v)$. Vertices in $new(K)$ numbered lower than v also belong to $ladj(v)$. Since no other cliques contain v, these are the only vertices in $ladj(v)$. ∎

A result of Schreiber [15] (see also Liu [12]) characterizes the structure of the v-th row of L as a pruned subtree of the etree rooted at v. This result now follows easily from the previous Theorem; the proof provides an intuitive understanding of why the row structure is a pruned subtree.

THEOREM 4.6. *The set $ladj(v) + v$ induces a pruned subtree $F(v)$ of the etree which is rooted at v.*

Proof. Let T be a clique tree of the graph G. Let v belong to $new(K)$, and as in Theorem 4.5, let $\mathcal{L} = \{L_1, \ldots, L_i\}$ be the lowest descendants of K in the clique tree T which contain v. A clique J which does not lie on a clique path from some clique in $\mathcal{L}$ to K does not contain v by Theorem 4.5, and thus any vertex in $new(J)$ cannot belong to $ladj(v)$.

The pruned subtree $F(v)$ can be characterized by its leaves. We can compute the leaves of the row subtree $F(v)$ by considering each clique path from some clique $L \in \mathcal{L}$ to K in T. Let L be any clique in the set $\mathcal{L}$, and write the cliques in the clique path from L to K in T as $D_j \equiv L$, D_{j-1}, ..., D_1, $D_0 \equiv K$. Let w_k denote the representative of the maximal clique D_k, where $0 \leq k \leq j$.

From Theorem 4.3, there is an etree path from each representative vertex w_k to v in the etree F. We can now describe the leaves of the row subtree $F(v)$ which correspond to the clique path from L to K. The vertex w_j is a leaf of $F(v)$. The other leaves (if any) of the row subtree corresponding to this clique path include all other representative vertices w_k, with $0 \leq k \leq j-1$, which do not lie on the etree path from w_j to v. By repeating this procedure for each clique $L \in \mathcal{L}$, we obtain all the leaves of the row subtree $F(v)$.

Consider the set of etree paths beginning at some leaf vertex belonging to the set of leaves defined above and ending at v. Each vertex reached by such a path belongs to a clique which includes v; further, such vertices are numbered lower than v by definition of the etree. It then follows that all such vertices belong to $ladj(v)$. ∎

The reader can verify that for the example we have considered, $ladj(10) = \{1,2,5,6,7,8,9\}$; the row subtree is the pruned subtree of the etree rooted at 10 with leaves 1 and 7; and that the cliques containing the vertex 10 form subtrees of both the clique trees we have obtained for this graph.

5. Compact and skeleton clique trees.

In the first subsection, we consider a compact data structure called the compact clique tree (cct) which can be computed from the clique tree. In the second subsection, we will consider a second compact data structure called the skeleton clique tree. Both these data structures are useful in applications where compact representations of the structure of the Cholesky factor are required.

5.1. Compact clique trees.

Let K_1, K_2, ..., K_m be the set of maximal cliques of a chordal filled graph G, and let T be a clique tree of G. Let K be a maximal clique whose set of child cliques is denoted by $\mathcal{C}(K)$. Recall that one property of a clique tree is that if C is a child of K, then $anc(C) \subset K$. Thus we can delete the vertices of K that belong to some child clique to obtain a space-efficient data structure. The maximal clique K may be recomputed when necessary from the anc sets of its children. Hence define the *compact clique* K^C corresponding to K as follows:

$$K^C = K \setminus \cup_{C \in \mathcal{C}(K)} anc(C).$$

Note that the compact clique K^C depends on the particular choice of the clique tree T used to represent G. The *compact clique tree* T^C is obtained from the clique tree T by replacing the maximal cliques by compact cliques for its vertices. If K is a leaf clique of the clique tree, then by definition, the compact clique is the clique itself.

Consider a maximal clique K and a vertex v belonging to $new(K)$. From the proof of Theorem 4.5, there is a set $\mathcal{L}$ of lowest descendant cliques of K such that v belongs to each clique $L \in \mathcal{L}$, and also to all cliques on the clique paths from each clique L to K. In the cct, v belongs only to the compact cliques which correspond to the cliques in $\mathcal{L}$, and not to any of the compact cliques of the other descendant cliques. Thus the cct could be potentially a much smaller data structure than the clique tree. Computational results on the sizes of the clique tree and cct for several problems in the Boeing-Harwell collection will be reported in Section 7.

As an illustration, we compute the ccts of the two clique trees in Figs. 2 and 5. In the first figure, the compact clique $K^C(5)$ has the empty set of vertices; all other cliques are leaf cliques, and hence unchanged. In the second figure, $K^C(5) = \{11\}$, $K^C(7) = \{7,8,9\}$, and the other cliques are unaffected.

We can compute the cct corresponding to a clique tree by a simple marking algorithm which marks all vertices of a maximal clique belonging also to the ancestor set of some child clique. Once such vertices are marked by considering all child cliques, the compact clique is computed to be the set of unmarked vertices in the maximal clique. Each maximal clique is examined at most three times: twice when its compact clique is being computed, and once when the compact clique of its parent is being computed. Thus the complexity of the algorithm for computing the cct from the clique tree is $O(q)$.

If we are given the cct T^C, then we can compute the maximal clique K by merging the ancestor sets of its child cliques with the compact clique K^C. If the maximal cliques are computed in post order on the cct, these ancestor sets will be available when we compute the maximal clique K. Since for each child clique C we have to decide if a vertex belongs to $new(C)$ or $anc(C)$, we maintain the *first_anc* vertex of each maximal clique. The clique tree can be computed from the cct in $O(q)$ time.

5.2. Skeleton clique trees. In this section, we consider another compact data structure for representing clique trees called the skeleton clique tree (sct). The skeleton clique tree relies on the concept of the skeleton matrix introduced by Liu [12]. Given the structure of a symmetric matrix A and an ordering for factoring the matrix, the skeleton matrix A^S is obtained by deleting every nonzero in A that would become a fill-nonzero under the given ordering of A. The skeleton graph G^S is the adjacency graph of the skeleton matrix A^S. Necessarily, the size (the number of nonzeros in the strict lower triangle) of the skeleton matrix is bounded by the size of the matrix A. We denote the size of the skeleton matrix by $|A^S|$. The skeleton graph of the chordal graph in Fig. 1 is shown in Fig. 6. The reader can verify that under the given ordering, all omitted edges are generated as fill edges.

In Theorem 4.6, we saw that the structure of the v-th row of L (the set $ladj(v)$ in G) is a pruned subtree of the etree. More precisely, the row subtree $F(v)$ is a subtree of the etree rooted at v, from which vertices have been pruned. The subtree $F(v)$ is characterized by identifying its leaves, and in the above Theorem, this was accomplished by means of a clique tree. The set of leaves of $F(v)$ is the set $ladj(v)$ in the skeleton graph G^S; hence this set is the row structure of the v-th row of the skeleton matrix A^S. The row structure of the v-th row of L (the set $ladj(v)$ in G) can then be generated from the set of leaves of $F(v)$ by traversing the etree path from each leaf to v, and including every vertex reached in the v-th

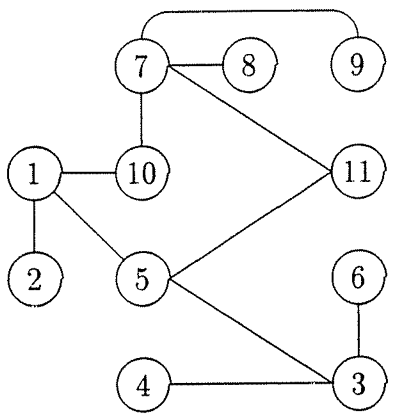

FIG. 6. *The skeleton graph of the chordal graph in Fig. 1. It can be verified that the corresponding filled graph is the chordal graph in Fig. 1.*

row of L. The row structures of A^S can be computed from A and the elimination tree in $O(|A|)$ time by an algorithm designed by Liu [12].

We define the skeleton higher adjacency set of a vertex v, $S(v)$, to be the set $hadj(v)$ in the skeleton graph G^S. These sets can be computed by a bucket sort of the $ladj(v)$ sets in the skeleton graphs in $O(|A^S|)$ time.

```
$i := 0$;
**for** $v := 1$ **to** $n \rightarrow$
    **if** $v$ has a child $u$ in etree **with** $hd(v) + 1 = hd(u)$ **then**
        $K_u := new\_in\_clique(u)$;
        $new\_in\_clique(v) := K_u$; $new(K_u) := new(K_u) + v$;
    **else**
        $i := i + 1$; $new\_in\_clique(v) := K_i$;
        **if** $v$ is a leaf in the etree **then**
            make $K_i^S := v + S(v)$ a skeleton clique;
        **else**
            make $K_i^S := S(v)$ a skeleton clique;
        **fi**
    **fi**
    **for each** child $s$ of $v$ in etree **with** $new\_in\_clique(s) \neq new\_in\_clique(v)$ **do**
        make $new\_in\_clique(s)$ a child of $new\_in\_clique(v)$ in the skeleton clique tree
    **rof**
**rof**
```

FIG. 7. *The Skeleton Clique Tree Algorithm*

We now define the sct T^S corresponding to a clique tree T. The sct has skeleton cliques for its vertices with parent relationships identical to those in the clique tree. The skeleton clique K^S corresponding to a maximal clique K is defined as follows. Let v be the representative vertex of K. We distinguish two cases: If K is a leaf clique in the clique tree, or if K

is an interior clique but v does not belong to any child clique of K, we define the skeleton clique $K^S = v + S(v)$. Otherwise, the skeleton clique $K^S = S(v)$. (The first case corresponds to v being a leaf in the elimination tree. The difference in the definitions in the two cases is slight, but it will be helpful to maintain the difference to make comparisons between the sct and the cct easier.)

In the clique tree in Fig. 2 the only non-leaf clique is the root clique; for this clique $K(5)$, the skeleton clique $K^S(5) = S(5) = \{11\}$. (The reader can verify that $S(5) = \{11\}$ from the skeleton graph in Fig. 6.) For each leaf clique, the skeleton clique is identical to itself. In the clique tree in Fig. 5, the interior cliques are $K(5)$ and $K(7)$. As before, the skeleton clique $K^S(5) = S(5) = \{11\}$. The skeleton clique $K^S(7) = K(7)$ since its representative vertex 7 is not contained in a child clique.

Next we design an algorithm, the Skeleton Clique Tree Algorithm, for computing the sct of a sparse, symmetric matrix. It requires the elimination tree, the higher degrees of the vertices, and the sets $S(v)$, the higher adjacency sets of vertices in the skeleton graph G^S. The computation of these data structures and degrees have been discussed earlier in Section 3 and in this subsection.

The Skeleton Clique Tree Algorithm is shown in Fig. 7. The only difference between this algorithm and the Clique Tree Algorithm 2 is that this algorithm computes skeleton cliques in addition to the *new* sets.

For the chordal graph in our examples so far, the Skeleton Clique Tree Algorithm can be used to compute scts corresponding to either of the two clique trees in Figs. 2 and 5. The particular sct computed depends on the choice of a child u of a vertex v satisfying the degree test in the algorithm.

6. Relationships between skeleton and compact clique trees.

Let T be any clique tree of a chordal filled graph G obtained by the Clique Tree Algorithm 2, and for each maximal clique K, let K^S, K^C denote the corresponding skeleton clique and compact clique with respect to T, respectively. In this section, we explore the relationships between the skeleton and compact cliques.

In this section we denote the representative vertex of a clique K by $rep(K)$. Let G^S denote the skeleton graph of the filled graph G, and let the higher adjacency set of a vertex v in G^S be denoted by $S(v)$. Note that $w \in S(v)$ implies that $v \in ladj(w)$ in the skeleton graph; i.e., v is a leaf of the row subtree $F(w)$. We make use of this fact to prove several results in this Section.

Our first result concerns a leaf clique.

LEMMA 6.1. *Let K be a leaf clique of a clique tree T with corresponding skeleton clique K^S and compact clique K^C. If $rep(K) = v$, then K, K^C, and K^S are all equal to $v + S(v)$.*

Proof. Since v is the representative vertex of K, $K = v + hadj(v)$. Since K has no child cliques, by definition $K^C = K$. Next we prove that $K^S = K$. Since K is a leaf clique, v is a leaf in the etree, and by definition $K^S = v + S(v)$. If a vertex w is distinct from v and belongs to K, then (v, w) is an edge in the graph G. The existence of this edge and the fact that v is a leaf of the etree together imply that v is a leaf of the row subtree $F(w)$. Thus w belongs to $S(v)$, hence to K^S, and thus $K \subseteq K^S$. Conversely, since $K^S = v + S(v)$ is a subset of the vertices in $v + hadj(v)$, we have $K^S \subseteq K$. Thus $K = K^S$. ■

Now we consider interior cliques in the clique tree. We begin by classifying the interior cliques into two groups. We denote by $\mathcal{C}(K)$ the set of child cliques of a maximal clique K, and define

$$\begin{aligned} low(K) &= \text{lowest-numbered vertex } v \text{ belonging to both } K \text{ and some child clique of } K \\ &= \min_{C \in \mathcal{C}(K)} \mathit{first\_anc}(C). \end{aligned}$$

Using the definition of $low(K)$, the interior cliques can be classified into two groups as follows:

$$\begin{aligned} \mathcal{I}_1 &= \{K\colon K \text{ is an interior clique with } low(K) > rep(K)\} \\ \mathcal{I}_2 &= \{K\colon K \text{ is an interior clique with } low(K) = rep(K)\}. \end{aligned}$$

First we consider the interior cliques in the group $\mathcal{I}_1$.

LEMMA 6.2. *If $K = \{w_0 < w_1 < \ldots < w_p\}$ is a maximal clique with $low(K) > w_0$, then $K^S = K$.*

Proof. First, we show that if $low(K) > w_0$, then w_0 is a leaf of the etree. By Lemma 4.1, vertices in $new(K)$ form a path in the etree from the lowest to the highest-numbered vertex. Since $low(K) > w_0$, and the lowest-numbered *new* vertex in K is w_0, the vertex w_0 is not contained in any child clique of K. Further, since w_0 is a *new* vertex in K, it may belong to other cliques only if they are descendants of K in T. From Theorem 7 in [10] (see the discussion about the properties of clique trees in Section 2), if w_0 belongs to some proper descendant D of K, it also belongs to the child C which lies on the clique path from D to K in T. Since w_0 does not belong to any child clique of K, it cannot belong to any other clique of T either. Thus w_0 belongs only to the maximal clique K, and hence it is a leaf of the etree.

Since K is a clique, it follows that w_0 is a leaf of each row subtree $F(w_j)$, for $1 \le j \le p$. Hence every vertex in K (other than w_0) belongs to $S(w_0)$, and thus $w_0 + S(w_0) \supseteq K$. Since w_0 is a leaf of the etree, the skeleton clique $K^S = w_0 + S(w_0)$, and hence $K^S \supseteq K$. As in Lemma 6.1, since $S(w_0)$ is a subset of the vertices in $hadj(w_0)$, we have $K^S = w_0 + S(w_0) = K$. ∎

From its definition, $K^C = K \setminus \cup_{C \in \mathcal{C}(K)} anc(C)$. Thus in this case, the difference between K^S and K^C is the union of the ancestor sets of the child cliques of K. The clique tree in Fig. 5 illustrates this case. Note that the clique $K = K(7) = \{7, 8, 9, 10, 11\}$ has $low(K) = 10$, which is greater than its representative vertex 7; hence the skeleton clique is the clique itself, while the compact clique $K^C = \{7, 8, 9\}$.

We now consider interior cliques in the group $\mathcal{I}_2$; i.e., the case when $low(K) = rep(K)$. Partition the child cliques of K into two sets:

$$\begin{aligned} \mathcal{C}_1 &= \{C \in \mathcal{C}(K) : \mathit{first\_anc}(C) = rep(K)\}, \\ \mathcal{C}_2 &= \{C \in \mathcal{C}(K) : \mathit{first\_anc}(C) > rep(K)\}. \end{aligned}$$

We will show that in this case the skeleton clique K^S is obtained by adding an *excess set* of vertices $E(K)$ to the compact clique K^C. Thus $K^S = K^C \cup E(K)$. We define $E(K)$ as the set of vertices $w_j \in K$ which belong to some clique in $\mathcal{C}_2$ but not to any clique in $\mathcal{C}_1$. The set of cliques $\mathcal{C}_1$ cannot be empty, since by assumption, $low(K) = w_0$; however, the set $\mathcal{C}_2$ can be empty. Then $E(K) = \emptyset$, and $K^C = K^S$.

The clique tree in Fig. 2 illustrates this case. Consider the clique $K = K(5) = \{5, 6, 10, 11\}$. The child cliques $K(1)$ and $K(3)$ belong to the set $\mathcal{C}_1$, and the clique $K(7)$ belongs to the set $\mathcal{C}_2$. The only ancestor vertex in the child cliques belonging to a clique in $\mathcal{C}_2$ but not to any clique in $\mathcal{C}_1$ is the vertex 11; hence $E(K) = \{11\}$. Note that the skeleton clique $K^S = \{11\}$, and that the compact clique $K^C = \emptyset$.

LEMMA 6.3. *If K is an interior clique with $low(K) = rep(K)$, then $K^C \cap E(K) = \emptyset$.*

Proof. Let the set of child cliques of K be partitioned into $\mathcal{C}_1$, $\mathcal{C}_2$ as above. If w_j belongs to K^C, then w_j does not belong to any child C, and hence not to $E(K)$. If w_j belongs to $E(K)$, then it belongs to some child clique $C \in \mathcal{C}_2$, and hence not to K^C. ■

LEMMA 6.4. *If $K = \{w_0 < w_1 < \ldots < w_p\}$ is an interior clique with $low(K) = w_0$, then $K^S = K^C \cup E(K)$.*

Proof. Since K is an interior clique, from its definition, $K^S = S(w_0)$. First we will show that $S(w_0) \supseteq K^C \cup E(K)$, and second, that the reverse containment holds, thus proving the lemma.

Suppose that $w_j \in K^C$, for some $1 \leq j \leq p$. From the definition of the skeleton graph of G, the edge (w_0, w_j) in G can be generated by a traversal of the row subtree $F(w_j)$ starting from a leaf vertex l of $F(w_j)$. The leaf l has the following properties: $l \leq w_0$, (l, w_j) is an edge in G, and w_0 is included on the etree path from l to w_j. Suppose for a contradiction that $l \neq w_0$. Let l belong to $new(C_i)$. By Theorem 4.3, corresponding to the $l - w_0$ path in the etree, there is a clique path C_i, C_{i-1}, …, C_1, K with $l \in new(C_i)$, and $w_0 \in new(K)$. Since (l, w_j) is an edge of G, $w_j \in hadj(l)$; since $l \in new(C_i)$, we have $hadj(l) \subset C_i$, and thus $w_j \in C_i$. Further, since w_j also belongs to the ancestor clique K of C_i, $w_j \in anc(C_i)$. But since w_j belongs to K, it also belongs to all cliques on the clique path from C_i to K. In particular, it belongs to C_1, a child of K. But then w_j cannot belong to K^C, contrary to assumption. This contradiction proves that $l = w_0$. Since w_0 is a leaf of the row subtree $F(w_j)$, we have $w_j \in S(w_0)$.

Now assume that $w_j \in E(K)$. As in the previous paragraph, since (w_0, w_j) is an edge in G, there is a leaf l of the row subtree $F(w_j)$ such that $l \leq w_0$, (l, w_j) is an edge of G, and w_0 is included on the etree path from l to w_j. Again, suppose $l \neq w_0$ to obtain a contradiction. Let l belong to $new(C_i)$. Corresponding to the $l - w_0$ path in the etree, by Theorem 4.3, there is a clique path C_i, C_{i-1}, …, C_1, K with $l \in new(C_i)$, $w_0 \in new(K)$, and $first\_anc(C_1) \leq w_0$. But $w_0 = low(K) = \min_{C \in \mathcal{C}(K)} first\_anc(C) \leq first\_anc(C_1)$, and thus we have $w_0 = first\_anc(C_1)$. Hence the clique C_1 belongs to the set $\mathcal{C}_1$. As in the previous paragraph, since $l \in new(C_i)$, $hadj(l) \subset C_i$; and since $w_j \in hadj(l)$, w_j belongs to C_i; indeed, w_j belongs to $anc(C_i)$, since it also belongs to the ancestor clique K. As before, w_j belongs to all cliques on the clique path from C_i to K, and in particular, to the set $anc(C_l)$. Now since $C_1 \in \mathcal{C}_1$, from the definition of the set $E(K)$, w_j cannot belong to $E(K)$. This contradicts the initial assumption that w_j belongs to $E(K)$, and proves that $l = w_0$. Thus we have $w_j \in S(w_0)$. This completes the proof that $S(w_0) \supseteq K^C \cup E(K)$.

We now prove that the reverse containment holds, thus proving the lemma. Since $S(w_0)$

is a subset of $hadj(w_0)$, a vertex in $S(w_0)$ belongs to $K = w_0 + hadj(w_0)$. Recall our notation $K = \{w_0 < w_1 < \ldots < w_p\}$, and let $w_j \in S(w_0)$. Then w_0 is a leaf of the row subtree $F(w_j)$. First, we claim that w_j does not belong to any clique in $\mathcal{C}_1$. To obtain a contradiction, suppose that w_j belongs to $C \in \mathcal{C}_1$, and let u be the representative vertex of C. Since $C \in \mathcal{C}_1$, we have $w_0 = \mathit{first\_anc}(C)$, and then $u < w_0$.

Now there are two cases to consider. If $w_j \in new(K)$, then by Lemma 4.2, there is an etree path from u to w_j which includes w_0. If $w_j \in anc(K)$, let w_j belong to $new(A)$, where A is an ancestor clique of K. Hence there is a clique path from C to A such that $u \in new(C)$ and $w_j \in new(A)$. Since $u \in new(C)$ and $w_0 = \mathit{first\_anc}(C)$, by Corollary 4.4, there is an etree path from u to w_j which includes w_0.

From the previous paragraph, there is an etree path from u to w_j which includes w_0; further, (u, w_j) and (w_0, w_j) are edges of G, the former because the vertices u, w_j belong to the clique C, the latter because the vertices w_0, w_j belong to K. But the (u, w_0) edge will generate the edges (x, w_0) for every vertex x on the etree path from u to w_j. Hence w_0 cannot be a leaf of the row subtree $F(w_j)$, contradicting assumption. This proves the claim.

Next, consider the set of vertices $X = \cup_{C \in \mathcal{C}_2} C$. Let $w_j \in S(w_0)$ be the vertex considered in the previous paragraphs. If w_j belongs to X, then it belongs to $E(K)$, since w_j does not belong to any clique in $\mathcal{C}_1$ by the claim. If w_j does not belong to X, then again from the claim, w_j belongs to no child of K, and hence it is included in K^C. Thus $K^C \cup E(K) \supseteq S(w_0)$. ■

We can now compare the sizes of Compact and Skeleton clique trees by means of the above results.

THEOREM 6.5.

$$|\text{sct}| = |\text{cct}| + \sum_{K \in \mathcal{I}_1} (|K \setminus K^C|) + \sum_{K \in \mathcal{I}_2} |E(K)|.$$

Proof. Immediate by summing over all the vertices of the cct and sct, using Lemmas 6.1, 6.2, and 6.4. ■

Let l_e denote the number of leaf vertices of the etree of the matrix A under a given ordering. Recall that the size of a symmetric matrix is the number of nonzeros in the strict lower triangle of that matrix.

COROLLARY 6.6.

$$|\text{cct}| \leq |\text{sct}| \leq \min\{q, |A| + l_e\}.$$

Proof. The first inequality follows from Theorem 6.5. The two terms in the right-hand side of the second inequality are obtained as follows.

Each leaf v of the etree is a representative vertex of some maximal clique K for which $K^S = v + S(v)$. For every other maximal clique K with representative vertex v, $K^S = S(v)$. Thus the size of the sct is bounded by the sum of the size of the skeleton matrix A^S and the number of leaves in the etree of A. The size of the skeleton matrix is no bigger than the size of the matrix A. This accounts for one term in the right-hand side.

The second term is obtained by noting that both the cct and the sct contain a subset of the vertices in each maximal clique of a clique tree T. ■

Grid graphs. Liu [12] shows that the skeleton graph of a $k \times k$ regular nine-point grid graph ordered by theoretical nested dissection ordering has no more than $2k^2$ edges. It is of interest to characterize ccts and scts for this model problem. For simplicity, we consider $k \times k$ nine-pont grids with $k = 2^m - 1$, where m is a positive integer. (Other values of k will have additional lower order terms in the size of the cct.)

THEOREM 6.7. *Let m be a positive integer, and let $k = 2^m - 1$. Consider a $k \times k$ nine-point regular grid graph ordered by theoretical nested dissection ordering. This graph has a cct of size $2k^2 + ((k+1)^2/4) - 2k$. The compact cliques corresponding to interior cliques are empty. Further, the sct is identical to the cct for this model problem.*

The second term in the size of the cct in the theorem is due to the leaf cliques in a clique tree, since in the compact clique corresponding to each leaf clique, we have included the representative vertex of the clique. This is solely for the purpose of ease in describing the cct, since the representative vertices of the maximal cliques would be stored separately in an actual realization of the cct data structure. The theorem is proved by solving a recurrence equation for the size of the cct in terms of the $(k-1)/2$ subgrids obtained when a '+' shaped separator is removed from the graph. We omit the proof.

A worst-case example. We now describe a class of graphs for which the size of the cct is $\Theta(n)$ whereas the sizes of the clique tree and sct are $\Theta(n^2)$. The graphs arise from the 'arrowhead matrix', and consist of an even number of vertices. Half the vertices (numbered $(n/2) + 1, \ldots, n$) form a clique; vertex 1 is connected to vertex n; vertex 2 to $n - 1$, n; and so on, till finally vertex $n/2$ is connected to all higher numbered vertices. The following facts are not difficult to verify: this graph has $n/2$ maximal cliques, and there is a clique tree of height $(n/2) - 1$; this clique tree has size $(n^2/8) + O(n)$; the corresponding cct has size n (each vertex appears exactly once); and the sct is identical to the clique tree. The large difference between the cct and the sct is caused by the fact that except for the single leaf clique, every other clique K belongs to the class $\mathcal{I}_1$; thus $K^S = K$, while K^C consists of a single vertex.

The results in the next Section will show that for several 'real' problems from the Boeing-Harwell collection, ordered by the minimum degree or nested dissection heuristics, the sizes of cct and sct are quite close. This can be attributed to the following observations. First, the number of cliques belonging to the class $\mathcal{I}_1$ is very small when these ordering algorithms are used to order A. Second, most interior cliques in the clique tree have few children (less than three on the average); and a vertex which belongs to the (one or two) child cliques in the class $\mathcal{C}_2$ is quite likely to belong also to the (one or two) child cliques in the class $\mathcal{C}_1$. Thus for most cliques K belonging to class $\mathcal{I}_2$, the set $E(K)$ is empty. The reasons why the computed clique trees have the above properties is a matter worthy of further study.

7. Computational results.

In this section, we report results pertaining to the computation of the various clique tree data structures for a set of large problems from the Boeing-Harwell collection [4]. The results we report were obtained by ordering the matrices using Liu's Multiple Minimum Degree (MMD) ordering [11]. All programs were written in C, and the computations were performed on one processor of a Cray-2.

The sizes of eight problems and the various data structures associated with them are shown in Table 1. The problems arise in different application areas as power networks, finite

TABLE 1
Sizes of problems and clique trees (scaled by 10^{-3})

key	n	$\|A\|$	$\|L\|$	$\|\mathrm{CT}\|$	$\|\mathrm{CCT}\|$	$\|\mathrm{SCT}\|$	$\|\mathrm{CCT}\|/n$
BCSPWR10	5.3	8.3	23	22	9.2	9.3	1.7
GRID 80 9	6.4	25	183	48	14	14	2.2
NASA2910	2.9	86	201	25	8.4	8.4	2.9
BCSSTK16	4.9	140	740	51	12	12	2.4
BCSSTK29	14	300	1,700	140	32	32	2.3
BCSSTK31	35.6	570	5,300	340	79	79	2.2
BCSSTK32	44.6	990	5,200	380	96	96	2.2
NASARB	54.9	1,300	11,900	600	141	142	2.6

element formulations of partial differential equations, fluid flow, and structural analysis. All numbers in the table are scaled by 10^{-3} for convenience. Recall that the size of a symmetric matrix is the number of nonzeros in its strict lower triangle. The number of nonzeros in L for these problems is about three to ten times the number of nonzeros in A.

It can be seen from the table that for all problems except the BCSPWR10 problem, the size of the clique tree is substantially smaller than the size of L. On the average problem, the clique tree is more than twelve times smaller than L, and the advantage increases with problem size. The power problem is typical of matrices with very high sparsity: the average column in A has only about 1.5 nonzeros in BCSPWR10. Such matrices incur low fill in L, have a large number of maximal cliques, and have a small average number of vertices per maximal clique. Hence the advantage of the clique tree representation over the adjacency list representation is small for such problems. However, this is not the case for problems in structural analysis, fluid flow, and finite element formulations of PDEs. Matrices from these areas have a relatively large number of nonzeros per column, consequently higher fill in L, and the clique tree representation offers large reduction in size over adjacency list representations. Indeed, for all other problems belonging to this class with the exception of the grid, the clique tree requires about half the size of the storage needed for the initial matrix A.

The sizes of the cct and the sct are almost the same for all these problems. We explained this in Section 6 by characterizing the two situations where the compact cliques and the skeleton cliques differ, and observing that these situations do not occur frequently when the Multiple Minimum Degree or Automated Nested Dissection orderings are used. We do not understand the reasons for this well.

The cct and sct data structures further reduce the storage required for the structure of L. The size of these compact data structures is about a quarter of the size of the clique tree for the average problem in the table. Perhaps the most significant statistic is the ratio of the size of the cct (sct) and the number of columns n. This number ranges from 1.7 to 2.9 for the problems in the table. The significance of this ratio is that it shows that the structure of L can be compactly stored in storage bounded by $3n$ for these problems.

In Table 2, we report the times required by the different steps in computing the cct and the sct from A, and the time to generate the clique tree from the sct. The reported

TABLE 2
Times for computing cct and sct and symbolic factorization (in units of 10^{-2} *sec on CRAY-2)*

key	$A \to L$	$L \to$ CT	CT $\to$ CCT	$A \to$ SCT	SCT $\to$ CT
BCSPWR10	2	1	1	7	1
GRID 80 9	16	2	10	59	6
NASA2910	57	1	5	125	3
BCSSTK16	100	1	10	230	6
BCSSTK29	190	3	28	480	17
BCSSTK31	320	8	66	1100	39
BCSSTK32	600	9	73	1500	43
NASARB	860	11	115	2500	66

times have been scaled for convenience; the unit of time is 10^{-2} seconds on a Cray-2. The computation of the cct requires a symbolic factorization step, the computation of the clique tree from L using the Clique Tree Algorithm 1, and the subsequent computation of the cct from the clique tree. The times required by these three steps are shown in the second, third, and fourth columns in the table. Note that the symbolic factorization step dominates the other two steps.

The sct is obtained by computing the elimination tree, the higher degrees, the skeleton graph, and the sorting required to obtain the higher adjacency lists $S(v)$. Then the Skeleton Clique Tree Algorithm is used to compute the sct. The total time for all these steps is shown in the fifth column of the table. Note that the computation of the sct requires more than twice the time required to compute the cct for these problems. It may be possible to design faster algorithms for computing the sct directly from A without computing the skeleton graph first. In any case, these times are quite small compared to the time requirements of the numerical factorization step.

Since the cct and the sct are quite compact, we should expect that it should be more efficient to generate the clique tree from them compared to symbolic factorization which generates L from A, since there is less merging to perform in the former case. The final column in Table 2 shows the time required to compute the clique tree from the sct. (Computing the clique tree from the cct takes similar amounts of time.) The second column shows the time taken by symbolic factorization to compute L from A. As expected, generating the clique tree from the sct is about eleven times faster than symbolic factorization on the average problem in the table.

8. Concluding remarks.

In this section, we briefly sketch some applications of the cct and the sct data structures, discuss related work, and mention some of the interesting questions raised by this work.

Applications. The cct and the sct can be used to organize an out-of-core multifrontal algorithm which is efficient with respect to storage. We consider one-pass algorithms in which once data is moved out of core, it is not required again. This feature of the algorithm reduces input-output operations which are typically expensive. A description of out-of-core algorithms for sparse Cholesky factorization may be found in Ashcraft et al. [2].

Initially consider using the clique tree to organize the algorithm. In the numerical factorization step, the columns of L corresponding to the *new* vertices of a maximal clique K can be computed together. The $anc(K)$ vertices form the row and column subscripts of the update submatrix to be stacked. We will refer to the computation of the columns of L in the set $new(K)$ as the computation of the clique K.

The numerical factorization is computed in post-order on the clique tree. When all the child cliques of K have been computed, the nonzeros in A belonging to the column set $new(K)$ are brought into core storage. The corresponding columns of L are then computed and stored out of core, and the update matrix stacked. The space used by these columns in the clique K can then be freed and used for other purposes.

The cct or the sct can be used to organize this algorithm instead of the clique tree, further reducing the integer overhead storage required. Notice that the vertex list of a maximal clique K is required only immediately before the nonzeros in columns of $new(K)$ are being computed. This list can be generated from the compact clique and the *anc* lists of its child cliques. The latter lists are available when K is being computed, since by then all of its child cliques have been computed. Thus the use of the cct or sct makes more space available to perform the factorization, reducing the number of garbage collection steps required in the course of the algorithm.

Ashcraft et al. [2] report that garbage collection on the integer subscripts together with the numerical values of the nonzeros provides an order of magnitude reduction in the integer storage required for an out-of-core sparse Cholesky factorization algorithm. They recommend that this feature be included in software for out-of-core factorizations.

The cct and sct can also be used to design storage-efficient algorithms for triangular solutions, when solutions corresponding to several right-hand side vectors are desired. Again, the core storage may be insufficient to hold the factor matrices and all the right-hand side vectors. Algorithms can be designed which make use of these compact data structures to reduce the primary and overhead storage required.

Suppose that the triangular factors of sparse, symmetric, positive definite matrix have been computed. The partial refactorization problem is the problem of computing a new factorization of the matrix when only a subset of the nonzero values change. The cct has the advantage that it exhibits clearly those columns whose nonzero values are changed.

Another application of these compact data structures is in the parallel computation of the factorization on a distributed-memory multiprocessor. Currently such architectures take longer to communicate a word of data between two processors than to perform a floating point operation on that word. Hence it will be advantageous in parallel algorithms to communicate the cct or sct between processors and then let each processor compute the clique tree.

Related work. The compressed column storage scheme of Sherman [16] is used in modern sparse matrix software. This is a column oriented storage scheme for the Cholesky factor in which overlaps between the row subscripts of consecutively numbered columns are exploited to compress the subscripts. In practice, the success of this scheme relies on an ordering algorithm for first identifying, and then numbering consecutively, sets of nodes which are indistinguishable with respect to elimination. The minimum degree ordering algorithms [7] identify such indistinguishable nodes by a heuristic method. The spectacular compression observed is primarily due to the underlying clique tree structure of the filled graph. A set of indistinguishable nodes corresponds roughly to a set of *new* nodes in a maximal clique of a clique tree.

Schreiber [15] has proposed an implementation of sparse Cholesky factorization where relative (rather than absolute) row subscripts are stored. This storage scheme makes use of the observation that the structure of a column j (except the diagonal element) is contained in the structure of its parent p in the elimination tree. The structure of column j is stored implicitly by storing the locations of the corresponding row subscripts in column p. Similarly, the structure of column p can be stored relative to its parent in the etree, and so on. These relative row subscripts can also be compressed as in the previous scheme.

Liu [12] has reported a compact row-oriented storage scheme which makes use of the skeleton matrix. The leaves of the row subtree $F(v)$ correspond to the structure of the v-th row of the skeleton matrix. Only these leaves are stored, and the structure of the v-th row of L can be computed from a traversal of the elimination tree from each leaf to v. George and Liu [8] have extended this work to static storage implementations of Gaussian elimination and orthogonal factorization.

Bank and Smith [3] have described a row-oriented data structure for the factorization of a sparse matrix with a symmetric nonzero structure in which overhead storage is required only for the nonzeros of A. The structure of the rows of L are computed as necessary from the corresponding rows of A. The overhead storage in this scheme could be further reduced by using the skeleton matrix introduced by Liu. However, this approach has a difficulty: the computation of the nonzero values of the k-th row requires the intersections of all rows $j < k$ such that l_{kj} is nonzero with row k. But the structure of such a row j is not necessarily a subset of the structure of row k. Hence it is not known how to compute these intersections in time proportional to the number of nonzeros involved in the computations at this step.

Eisenstat, Schultz, and Sherman [5] have proposed a variant of Cholesky factorization for finite element problems on machines with limited core storage without making use of auxiliary storage. Their idea is to discard most nonzero values in the columns of L, and recompute them as necessary within the framework of a divide and conquer approach. The algorithm is organized around a data structure called the element merge tree, which is similar to the elimination tree.

Ashcraft [1] has reported results from a vectorized implementation of a multifrontal sparse Cholesky factorization. He computes a supernodal elimination tree which is defined on the supernodes; the supernodes roughly correspond to the *new* sets of the maximal cliques in a clique tree. Here, in the symbolic factorization step, the structure of a supernode is computed by merging the row subscripts of the children of the supernode with the subscripts of the columns of A which belong to the supernode. Ashcraft reports that not much effort was put into optimizing this computation, and that this step took roughly the time taken by a symbolic factorization routine in Sparspak— about a quarter to half the time of the numerical factorization step on a single processor of a Cray X-MP.

The above supernodal symbolic factorization is similar to the computation of a clique tree from the cct in that both compute the structure of a group of columns from the structures of other groups. The major differences are that the cct does not require the columns of A, and that much less merging is required with the cct because it has fewer redundant subscripts to merge, being a more compact data structure. Indeed, the results in § 7 show that the computation of the clique tree from the sct is on the average about eleven times faster than symbolic factorization on the problems considered there.

Future work. In the discussion of applications, we have considered some of the algorithmic contexts in which the cct and the sct are useful data structures. It would be helpful to find other contexts where these data structures would be useful. The clique tree, cct, and sct data structures could also be extended to the contexts of sparse Gaussian elimination and sparse orthogonal factorization, in a manner similar to the work in George and Liu [8].

Various factors that influence the size of ccts and scts are also worthy of further study. The set of maximal cliques of a chordal graph is independent of the ordering of the vertices. Hence the size of a clique tree is unchanged when the vertices are reordered by another peo. However, the situation is different for compact clique trees. Note that several clique trees with vastly different structures could be computed from the same peo, and that a cct is defined with respect to a particular clique tree. Thus the size of a cct depends on the clique tree with respect to which it is defined. Indeed, there are worst-case examples of chordal graphs such that it is possible to compute two compact clique trees corresponding to two different clique trees of the graph such that one cct has size $\Theta(n)$ while the other has size $\Theta(n^2)$. Thus a study of the influence of clique tree structure on the size of the cct seems useful.

The size of the sct is the sum of the size of the skeleton graph and the number of leaves in the etree. Hence the sct size is independent of the structure of the clique tree, and depends only on the peo of the vertices. However, the size of the skeleton graph does depend on the ordering, and little is known about this dependence.

The relationship between the cct and the sct remains to be fully understood. The experimental results in this paper show that with commonly used ordering algorithms, the sizes of these data structures are almost identical. Since we have characterized the situations in which the compact clique differs from the skeleton clique, the results imply that these situations do not occur frequently in practice. It would be worthwhile to understand this behavior of the ordering algorithms.

Another question of interest is whether we can compute the cct from the sct efficiently. It is possible to construct examples where this cannot be done solely by means of 'local operations'. By a local operation, we mean a comparison of a clique with its parent or any of its children.

More efficient algorithms for generating the sct from the matrix A would be desirable, since currently this computation is slower than the computation of the cct. It would also be nice to know if there exists an algorithm for computing the cct directly from A rather than from the clique tree.

REFERENCES

[1] C. Ashcraft, *A vector implementation of the multifrontal method for large, sparse, symmetric, positive definite linear systems*, Tech. Report ETA-TR-51, Engineering Technology Applications Divi-