This paper appears in
Large-Scale Numerical Optimization, T. F. Coleman and Y. Li, eds.,
SIAM, Philadelphia, 1990, pp. 180-204.

# 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 [16] 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. Matrix interpretations of these data structures are provided. The situations when these data structures differ are characterized, and it is proved that the compact clique tree is the smaller of the two in size. There exist filled graphs on $n$ vertices where the compact clique tree has size $\Theta(n)$ while the skeleton clique tree has size $\Theta(n^2)$, though experimentally the two are almost identical for filled graphs obtained from common ordering algorithms. A partial justification is provided for this phenomenon. Computational results on some Boeing-Harwell test problems show that these data structures require substantially less space than either the clique tree or the matrix $A$.

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

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

# 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 [16] 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. Matrix interpretations of these data structures are provided. The situations when these data structures differ are characterized, and it is proved that the compact clique tree is the smaller of the two in size. There exist filled graphs on $n$ vertices where the compact clique tree has size $\Theta(n)$ while the skeleton clique tree has size $\Theta(n^2)$, though experimentally the two are almost identical for filled graphs obtained from common ordering algorithms. A partial justification is provided for this phenomenon. Computational results on some Boeing-Harwell test problems show that these data structures require substantially less space than either the clique tree or the matrix $A$.

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

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

**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? Apart from its intrinsic theoretical interest, the answer to this question is potentially useful in the design of efficient out-of-core algorithms for the solution of sparse systems of linear equations. One such context arises in large unconstrained optimization problems when a large, sparse Hessian matrix must be factored at each iteration, with its nonzero structure remaining fixed. A space-efficient data structure for the structure of the Cholesky factor of the Hessian can be used to reduce the overall storage requirements of the algorithm. Since the compact data structure for the structure of the Cholesky factor need be computed only once, the costs of computing such a data structure can be amortized over the number of iterations in the optimization algorithm.

There are two versions in which the above 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 [16]. The existence of this data structure is the primary

* Presented at the Workshop on Large-Scale Numerical Optimization, Mathematical Sciences Institute, Cornell University, Oct 18–20, 1989. Written December 1989; revised April 1990.

† 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.

reason for the spectacular success of the *compressed column storage scheme* proposed by Sherman [23] 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 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 [18]. 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 [18] which makes use of the skeleton graph. The skeleton graph is defined from the perspective of individual vertices of the adjacency graph of $A$. We will argue that the proper way of generalizing the concept of the skeleton graph from the perspective of supernodes results in the compact clique tree.

An outline of this paper is as follows. In § 2, we describe an algorithm for computing a clique tree from the elimination tree, discuss relevant properties of clique trees, and consider the relationships between maximal cliques and supernodes. We consider the compact clique tree and the skeleton clique tree data structures, and provide matrix interpretations of these data structures in § 3. § 4 explores the relations between paths in clique trees and elimination trees. The results from this section are necessary to study the relationships between compact clique trees and skeleton clique trees. In § 5, we characterize the two 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, and prove that the compact clique tree is the smaller of the two in size. § 6 includes discussions of these data structures for two important classes of graphs, the grid graph and the arrowhead graph, and a result that partially justifies the fact that these data structures are experimentally close in size. 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 § 8 briefly discusses some applications, related work, and some questions raised by this research.

We discuss the notation employed in the next few paragraphs.

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 $G$, and the vertex set of $G$ by $V$. The number of nonzeros in the strict lower triangle of $A$ ($L$) will be denoted by $|A|$ ($|L|$).

It is well-known that $G$ is a chordal graph, i.e., every cycle in $G$ with more than three edges has a *chord*, an edge joining two vertices which are not adjacent in the cycle. We will assume some knowledge of terminology associated with sparse matrix factorizations and chordal graphs; the reader may find discussions of these in Liu [19] and Golumbic [14]. The vertices of a chordal graph can be ordered such that they can be eliminated in this order without causing any fill, and such an ordering is called a *perfect elimination ordering* (PEO).

A clique in $G$ is a subgraph of $G$ such that every pair of its vertices is adjacent. We will not distinguish between a clique and its set of vertices. A clique $K$ is a *maximal clique* if there exists no other vertex $v$ of $G$ such that $K \cup \{v\}$ is a clique. A result of Fulkerson and Gross [9] in the next section shows that the maximal cliques of a chordal graph are unique. If we let $K_1$, $K_2$, …, $K_m$ denote the maximal cliques of $G$, the number of maximal cliques is $m$, and the sum $q = \sum_{i=1}^{m} |K_i|$ is the total number of vertices in the maximal cliques. This number is also the size of a clique tree.

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. Clique trees.** In this section, we introduce the concept of a clique tree, describe an algorithm for computing a clique tree from the elimination tree, and discuss some properties of clique trees which we will require later. The clique tree algorithm will be needed to describe the skeleton clique tree data structure. One of the difficulties in describing clique trees is the lack of uniqueness: given a PEO of a chordal graph, while the elimination tree is unique, a clique tree is not necessarily unique. Another difficulty is that there have been two definitions of clique trees that have been used in the past, one being more restrictive than the other. We briefly consider several approaches for computing clique trees, and then indicate the approach we are going to take in this paper.

A general approach uses the *clique graph*, which is the edge-weighted intersection graph of the maximal cliques. The weight of an edge joining two maximal cliques $K_i$ and $K_j$ is the number of vertices common to both cliques. Blair, English, and Thomason [5] then show that a clique tree may be characterized as any maximum-weight spanning tree of the clique graph. Earlier approaches for computing clique trees include Gavril [10], Walter [25], and Tarjan and Yannakakis [24] (see also Golumbic [14]). In his thesis, Peyton [20] describes an $O(e)$ algorithm to compute clique trees from the adjacency lists of the chordal graph using maximum cardinality search.

In the context of sparse matrix factorizations, it is more efficient to work with the the higher adjacency lists of the vertices stored in a compressed format, since these lists are available as the output of a symbolic factorization algorithm. The paper [16] describes an $O(n)$ algorithm for computing a clique tree from these lists. The goal of this paper was to obtain an efficient algorithm for implementing the Jess and Kees method for reordering sparse matrices for parallel factorization, and to achieve this goal a more restrictive definition of a clique tree was required. We make the nature of this restriction precise later in this section.

In this paper, we describe an algorithm for computing a clique tree (more precisely, a subset of the information in the clique tree) directly from the elimination tree and the higher degrees, since it is helpful in describing the concept of a skeleton clique tree. The elimination tree and the higher degrees can be computed directly from $A$ without a symbolic factorization, and hence the clique tree algorithm presented here does not require the symbolic factorization step either. This clique tree algorithm is useful in several contexts where only a subset of the information in the clique tree is required. This algorithm also uses the more restrictive definition of a clique tree. Before we can describe this algorithm, we require some background material on elimination trees and the maximal clique structure of chordal graphs.

**Background**. 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. Since the etree captures the column dependencies in the Cholesky factorization of the matrix $A$, it plays an important role in modern sparse matrix algorithms. Liu [19] provides an excellent survey of this role. The etree can be computed in $O(|A|\alpha(|A|, n))$ time from the matrix $A$ by an algorithm of Zmijewski and Gilbert [26], where $\alpha(*, *)$ is the functional inverse of Ackermann's function. It is well-known (e.g., see [19]) that the higher degrees of the vertices can then be computed from $A$ and the etree in $O(|A|)$ space and $O(|L|)$ time.

A chordal graph numbered in a PEO is shown in Fig. 1. Its elimination tree and the higher degrees of the vertices are also shown.

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 [9].

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 $K = \{v\} \cup hadj(v)$.*

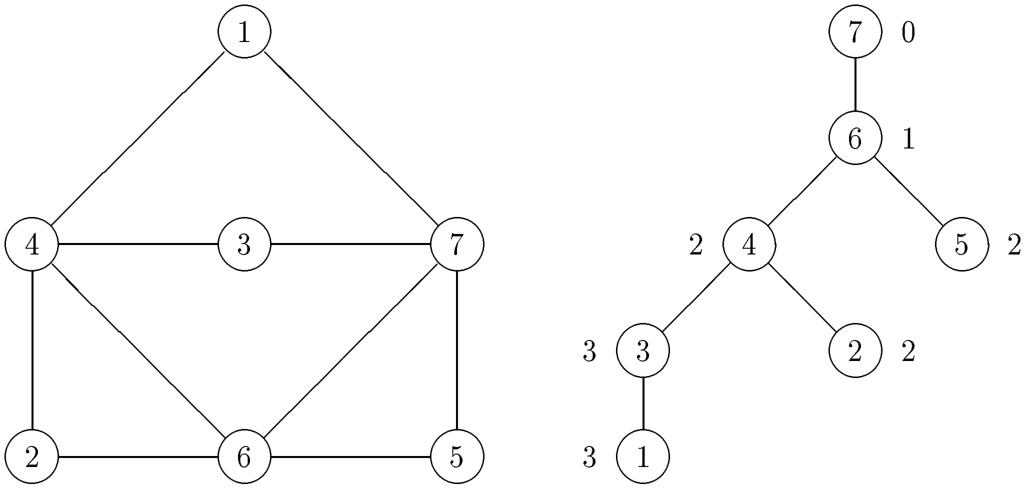


FIG. 1. *A chordal graph numbered in a perfect elimination ordering, and its elimination tree. Each internal face of this planar graph is a clique; such 'diagonal' edges have not been drawn for clarity. The higher degrees of the vertices are shown next to the vertices in the elimination tree.*

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 $v \cup hadj(v)$ generated by the lowest-numbered or *representative* vertex $v$. If a vertex is not the representative of *any* maximal clique, we call it a *nonrepresentative* vertex. From Proposition 2.1, it is only necessary to identify all representative vertices to obtain the maximal cliques of $G$. Note that the while the representative vertices depend on the PEO used, the maximal cliques do not since they are unique.

The reader can verify that the maximal cliques of the chordal graph in Figure 1 are as follows: $K(1) = \{1, 3, 4, 7\}$, $K(2) = \{2, 4, 6\}$, $K(3) = \{3, 4, 6, 7\}$, and $K(5) = \{5, 6, 7\}$.

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

PROPOSITION 2.2 ([16]). *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 $v \cup hadj(v) \subset z \cup hadj(z)$.*

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

PROPOSITION 2.3 ([16]). *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 $0 \leq j \leq i$. 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 $hadj(w_0) \subseteq \{w_1\} \cup hadj(w_1)$. By repeating this argument for $w_1$, …, $w_{p-1}$, we have

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

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

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

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

**A clique tree algorithm.** The above results can be used to design an algorithm for computing a clique tree of $G$ from the etree and the higher degrees. The algorithm identifies the representative vertices of the maximal cliques, and implicitly partitions each maximal clique $K$ into two ordered subsets of vertices called $new(K)$ (read 'new set of $K$') and $anc(K)$ (read 'ancestor set of $K$'). This partition is obtained in terms of a vertex called the $first\_anc(K)$ ('read first ancestor vertex of $K$'), such that all vertices in $K$ numbered lower than $first\_anc(K)$ belongs to $new(K)$, and the set $anc(K)$ includes $first\_anc(K)$ and vertices numbered higher than $first\_anc(K)$. It will turn out that the *new* sets of the maximal cliques of $G$ partition the vertices of $G$. Hence we can define mapping a $new\_in\_clique(v)$ for each vertex $v$ which is the unique clique in which $v$ is a *new* vertex. A clique $P$ is then defined to be the parent of a clique $K$ if $first\_anc(K)$ belongs to $new(P)$.

The Clique Tree Algorithm is shown in Fig. 2. The algorithm processes the vertices of the etree from 1 to $n$. If a vertex $v$ has a child $u$ in the etree with $1 + hd(v) = hd(u)$, then $v \cup hadj(v) \subset \{u\} \cup hadj(u)$, $v$ is not a representative vertex, and $v$ is made a *new* vertex in the clique $K_j$ 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 a clique $K_\ell$ as a *new* vertex, we make $K_\ell$ a child of $K_i$ in the clique tree.

$i := 0;$
**for** $v := 1$ **to** $n \rightarrow$
  **if** $v$ has a child $u$ in etree **with** $hd(v) + 1 = hd(u)$ **then**
    let $K_j$ be the clique which contains $u$ as a *new* vertex;
    $new\_in\_clique(v) := K_j$; $new(K_j) := new(K_j) \cup \{v\}$;
  **else**
    $i := i + 1;$
    make $v$ the representative of a maximal clique $K_i = v \cup hadj(v)$;
    $new\_in\_clique(v) := K_i$; $new(K_i) := \{v\}$;
  **fi**
  **for each** child $s$ of $v$ in etree **with** $new\_in\_clique(s) \neq new\_in\_clique(v) \rightarrow$
    let $K_\ell$ be the clique which contains $s$ as a *new* vertex;
    $first\_anc(K_\ell) := v$;
    make $K_\ell$ a child of $new\_in\_clique(v)$ in the clique tree;
  **rof**
**rof**

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

We can now describe how a maximal clique $K$ is partitioned by the Clique Tree Algorithm. For each maximal clique $K = \{w_0 < w_1 < \ldots < w_p\}$ (except the root clique), the algorithm obtains a vertex $first\_anc(K)$, the set $new(K)$, and implicitly the set $anc(K)$, such that

1. $new(K) = \{w_0 < w_1 < \ldots < w_{l-1}\}$, where $1 \leq l \leq p$, such that $hadj(w_0) = w_1 + hadj(w_1) = \ldots = \{w_0, w_1, \ldots, w_{l-1}\} \cup hadj(w_{l-1})$. Thus $hadj(w_j) \subset K$, for $1 \leq j \leq l-1$.
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.

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 such a child is included as a *new* vertex. Hence even though the maximal cliques of a chordal graph are unique, the partition of each maximal clique $K$ into the sets $new(K)$ and $anc(K)$ is not necessarily unique. Consequently, the clique tree is not necessarily unique, and this freedom can be used by the Clique Tree Algorithm to compute clique trees with differing properties.

The Clique Tree Algorithm computes only the $first\_anc(K)$ and $new(K)$ vertices explicitly; it does not obtain the *anc* sets of the maximal cliques. In some contexts, only the former information is required, and one such context arises in the next section where we compute skeleton clique trees. However, in other situations the *anc* sets are required explicitly. Suppose that the higher adjacency lists of the vertices are available from a symbolic factorization. If $w_0$ is the representative vertex of $K$, then $anc(K)$ is the set of vertices in the higher adjacency list of $w_0$ which are numbered higher than or equal to the vertex $first\_anc(K)$. If this list is sorted in increasing order, then a pointer to the $first\_anc(K)$ vertex is all that is necessary for maintaining the partition into $new(K)$ and $anc(K)$.

Two clique trees of the filled graph in Fig. 1 are shown in Fig. 3. Either clique tree can be computed by the Clique Tree Algorithm from the etree of that graph. The two different clique trees are obtained by choosing to merge vertex 6 into either the clique which includes its child 4 as a *new* vertex, or the clique which includes its other child 5 as a *new* vertex.

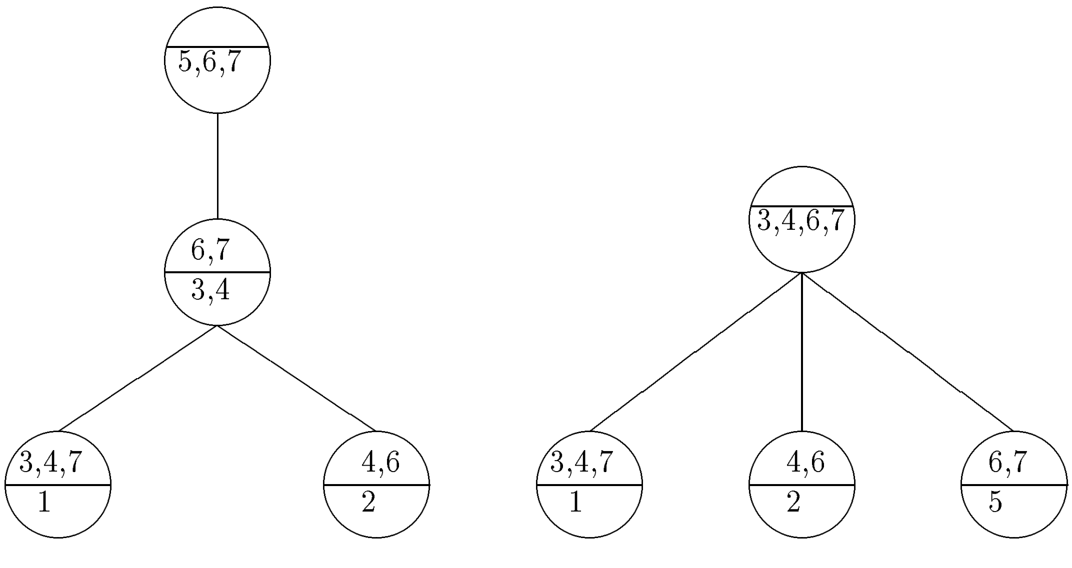


FIG. 3. *Two clique trees of the chordal graph in Fig. 1.*

The Clique Tree 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. We omit a proof of correctness of the Clique Tree Algorithm for brevity. From the following discussion of the properties of clique trees, it will turn out that the algorithm computes a correct 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. Let $P$ denote the parent clique of a maximal clique $K$ in a clique tree $T$. As a consequence of our definition of a clique tree, it can be proved (Proposition 6 and Theorem 7, [16]) 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 $K$;
- a vertex $w \in anc(K)$ belongs to the *new* set of some ancestor clique $J$; further, $w$ belongs to all cliques which lie on the path from $K$ to $J$ in the clique tree; and
- $P$ is the only ancestor clique of $K$ which contains all vertices in $anc(K)$; other ancestor cliques can contain only proper subsets of $anc(K)$.

The first three properties characterize 'legal' clique trees of chordal graphs. The fourth property makes our definition more restrictive, and is a consequence of the definition of the parent clique $P$ as the clique containing $first\_anc(K)$ as a *new* vertex. Another consequence of the second and third properties is that the set of cliques containing a vertex $w$ forms a subtree of the clique tree $T$.

It is instructive to consider the significance of the partition of each maximal clique $K$ into the $new(K)$ and $anc(K)$ sets with respect to a 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 in $K$ that also belong to some ancestor cliques of $K$, and the vertices in $new(K)$ have not appeared in any of the previously added cliques. In the less restrictive definition of a clique tree, a parent of $K$ can be chosen to be any clique which contains the set $anc(K)$, when more than one clique contains the set $anc(K)$. The cliques which are eligible to be the parent of $K$ form a rooted subtree of the partially formed clique tree. Our restrictive definition corresponds to choosing the root clique of this subtree as the parent of $K$.

**Supernodes and clique trees**. Supernodes have been used in the ordering and symbolic factorization steps [12] and more recently, in the numeric factorization step of sparse factorizations [2] to increase the efficiency of the computations. A *partial supernode* $S$ is a clique $\{s_1, \ldots, s_p\}$ in the filled graph $G$ such that $hadj(s_i) = \{s_{i+1}\} \cup hadj(s_{i+1})$, for $i = 1$, $\ldots$, $p-1$. A set of partial supernodes $S_1$, $\ldots$, $S_q$ is a *partial supernode partition* if they form a partition of the vertex set $V$. Let $S$ be a clique in a partial supernode partition, and let the parent vertex of $s_p$ in the etree be $t$. We call $S$ a supernode which is *maximal upto partition* (maximal supernode) (i). either if $hadj(s_p) \neq \{t\} \cup hadj(t)$, (ii). or if $hadj(s_p) = \{t\} \cup hadj(t)$, then there exists a child $u$ of $t$ such that $t$ belongs to a partial supernode containing $u$. Finally, a partial supernode partition is a *supernode partition* if every supernode in the partition is a maximal supernode.

Partial supernode partitions have been used by Ashcraft et al. [2] for vectorized numerical factorization. The paper [21] includes an $O(n)$ algorithm for computing the supernode partition from the etree. It is not difficult to verify that the *new* sets of the maximal cliques in a clique tree form a supernode partition. The connections between supernodes and clique trees are explored in greater detail in [15], where in particular it is shown that an algorithm for computing a supernode partition can be modified to compute a clique tree.

**3. Compact and skeleton clique trees.** This section is organized into three subsections: in the first, we consider the compact clique tree (cct) and its computation from the clique tree; in the second subsection, we consider the skeleton clique tree (sct); and in the final subsection, we consider matrix interpretations of the cct and the sct.

**3.1. Compact clique trees.** Let $\mathcal{C}(K)$ denote the set of child cliques of a maximal clique $K$. 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 properties of clique trees in § 2, the set of cliques containing $v$ forms a rooted subtree of $T$ with $K$ as the root clique. Hence there is a set $\mathcal{M}$ of 'lowest descendant cliques' of $K$ such that for each $M \in \mathcal{M}$, $v$ belongs to $M$ but not to any descendant of $M$, and $v$ belongs to all cliques on the clique path from $M$ to $K$. In the cct, $v$ belongs only to the compact cliques which correspond to the cliques in $\mathcal{M}$, and not to any other compact 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 § 7.

As an illustration, we compute the ccts of the two clique trees in Figs. 3. For the first clique tree, the compact clique $K^C(3) = \emptyset$, and $K^C(5) = \{5\}$; all others are leaf cliques, and hence unchanged. For the second clique tree, $K^C(3) = \emptyset$, and since the others are leaf 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 $O(q)$.

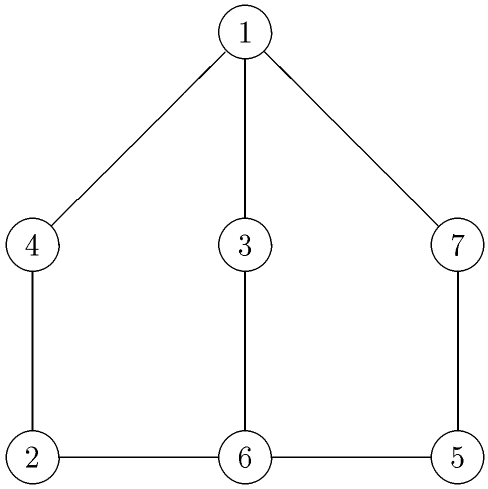


FIG. 4. *The skeleton graph of the chordal graph in Fig. 1.*

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.

**3.2. Skeleton clique trees.** The skeleton clique tree relies on the concept of the skeleton matrix introduced by Liu [18]. 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 $A^S$. Clearly 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. 4. The reader can verify that under the given ordering, all omitted edges are generated as fill edges.

A result of Schreiber [22] (see also Liu [18]) characterizes the structure of the $v$-th row of $L$ as a subtree $F(v)$ of the etree rooted at $v$ from which vertices have been pruned. The subtree $F(v)$ can be characterized by identifying its leaves; the set of leaves of $F(v)$ is the set $ladj(v)$ in the skeleton graph $G^S$. Hence this set is the structure of the $v$-th row of the skeleton matrix $A^S$. The row structure of the $v$-th row of $L$ 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 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 [18]. 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.

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 $K$ with representative vertex $v$ is defined as follows. We distinguish two cases: If $K$ is a leaf clique in the clique tree, or if $K$ is an

```
i := 0;
for v := 1 to n →
    if v has a child u in etree with hd(v) + 1 = hd(u) then
        let K_j be the clique which contains u as a new vertex;
        new_in_clique(v) := K_j; new(K_j) := new(K_j) ∪ {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) ≠ 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. 5. *The Skeleton Clique Tree Algorithm*

interior clique but $v$ does not belong to any child clique of $K$, we define the skeleton clique $K^S = v \cup S(v)$. 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.)

Again, we use the clique trees in Fig. 3 for illustration. The reader can verify the following from the skeleton graph in Fig. 4. In the first clique tree, $K^S(3) = S(3) = \{6\}$, and $K^S(5) = K(5)$. The other cliques are leaves of the clique tree, and for these, the skeleton cliques are identical to the cliques. In the second clique tree in Fig. 3, the only non-leaf clique is the root clique $K(3)$; again, the skeleton clique $K^S(3) = S(3) = \{6\}$. The other cliques are leaf cliques, and as before, for each leaf clique, the skeleton clique is identical to itself.

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 Skeleton Clique Tree Algorithm is shown in Fig. 5. The only difference between this algorithm and the Clique Tree Algorithm is that this algorithm computes skeleton cliques in addition to the parent relationships in the clique tree and 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 Fig. 3.

We have described the sct using the Skeleton Clique Tree Algorithm for the sake of concreteness. It should be clear now that the sct can be computed from any clique tree $T$, not necessarily obtained from the Skeleton Clique Tree Algorithm. To obtain the sct

$$L_S = \begin{pmatrix} 1 & & & & & & \\ \cdot & 2 & & & & & \\ \times & \cdot & 3 & & & & \\ \times & \times & 4 & & & & \\ \cdot & \cdot & \cdot & 5 & & & \\ \cdot & \times & \times & 6 & & & \\ \times & \cdot & \times & 7 & & & \end{pmatrix} \qquad A^{SS} = \begin{pmatrix} 1 & & & \\ \cdot & 2 & & \\ \times & \cdot & 3 & \\ \times & \times & 4 & \\ \cdot & \cdot & \cdot & 5 \\ \cdot & \times & \times & 6 \\ \times & \cdot & \cdot & 7 \end{pmatrix} \qquad A^{C} = \begin{pmatrix} 1 & & & \\ \cdot & 2 & & \\ \times & \cdot & 3 & \\ \times & \times & 4 & \\ \cdot & \cdot & \cdot & 5 \\ \cdot & \times & \cdot & 6 \\ \times & \cdot & \cdot & 7 \end{pmatrix}$$

FIG. 6. *The supernodal Cholesky factor, the supernodal skeleton matrix, and the compact matrix of the graph in Fig. 1.*

corresponding to a clique tree, we require only the parent relationships in the clique tree, the representative vertices of the maximal cliques, and the sets $S(v)$ for each representative vertex $v$.

**3.3. A matrix interpretation.** It will be helpful to consider matrix representations of the cct and the sct to understand these data structures better. We consider three matrix representations: a supernodal Cholesky factor, a supernodal version of the skeleton matrix to aid in the understanding of the sct, and a matrix called the *compact matrix* for the cct.

The *supernodal Cholesky factor* $L_S$ is a matrix whose columns are the maximal cliques of the filled graph $G$ and whose rows are the vertices of $G$. For each maximal clique $K$, we associate its column with the vertices in $new(K)$ (a supernode) by writing down the numbers of these vertices explicitly. The vertices in $anc(K)$ are marked by the '$\times$' symbol in the corresponding rows. The '$\cdot$' symbol in the column corresponding to $K$ indicates that the corresponding vertex is not a member of $K$. The supernodal Cholesky factor of the graph in Fig. 1 is shown in in Fig. 6. Note that the size of the clique tree is the number of nonzeros in this matrix representation.

The *supernodal skeleton matrix* $A^{SS}$ and the *compact matrix* $A^C$ have similar representations as the supernodal Cholesky factor, except that the former has skeleton cliques and the latter has compact cliques for columns. Again, we associate a column corresponding to the skeleton or compact clique of a clique $K$ with the vertices in $new(K)$, and indicate this by writing down the numbers of these vertices. Note that some of the vertices in $new(K)$ may not be included in the skeleton or compact clique; nevertheless, for the sake of clarity in the matrix representation, we list the numbers of all these vertices in the corresponding column. Since the *new* sets of all the maximal cliques together partition the vertices, this modification can increase the storage by at most $n$.

The supernodal skeleton matrix provides a supernodal representation of the skeleton matrix introduced by Liu [18]. The supernodal skeleton matrix of the graph in Fig. 1 is shown in Fig. 6.

We can consider computing the structure of the matrix $L_S$ from the supernodal skeleton matrix $A^{SS}$, by the elimination of columns. Consider the column of $A^{SS}$ which corresponds to the clique $K = K(3)$ with $new(K) = \{3, 4\}$ in Fig. 6. Since column 1 has nonzeros in rows 3, 4, and 7, the column corresponding to $K$ inherits the nonzero in row 7 during elimination.

Now column 2 has a nonzero in row 6, but since it does not have a nonzero in row 3, $K$ cannot inherit this nonzero, if we consider the fill in elimination from the perspective of individual columns. Thus the supernodal skeleton matrix $A^{SS}$ (and the skeleton matrix $A^S$) must include a nonzero in row 6 of column 3. In other words, the skeleton clique $K^S$ must contain the vertex 6.

Now we turn to the *compact matrix* $A^C$. The structure of the compact matrix corresponding to the graph in Fig. 1 is also shown in Fig. 6. Again, we consider the computation of the structure of the matrix $L_S$ from the compact matrix $A^C$. Think of the column corresponding to the clique $K = K(3)$ with $new(K) = \{3, 4\}$ whose child cliques are $K(1)$ and $K(2)$. Since column 1 has nonzeros in rows 3, 4 and 7, $K$ inherits the nonzero in row 7. Also, since column 2 has nonzeros in rows 4 and 6, and since column 4 belongs to $new(K)$, $K$ inherits the nonzero in row 6 from column 2. The difference here is that now we consider fill from the perspective of the supernodes or maximal cliques. Thus in the compact matrix $A^C$, column $K$ has zeros in rows 6 and 7. In other words, the compact clique $K^C$ is empty.

An important observation here is that a compact clique $K^C$ can inherit the nonzero structure of a child clique $C$ through a vertex $w$ in $new(K)$ other than the representative vertex of $K$, if $w$ also belongs to $C$. In the case of the sct, a skeleton clique can inherit the nonzero structures of only those child cliques $C$ which contain the representative vertex of $K$. This is because the sct does not fully exploit the structure of the clique tree, since it is defined using the skeleton matrix which is 'built up' by considering the fill-in process from a nodal perspective. On the other hand, the cct considers fill-in from the perspective of the maximal cliques (supernodes). We characterize the situations where the compact cliques and skeleton cliques differ from each other in § 5.

**4. Paths in clique trees and etrees.** In this section, we study the relationships between paths in etrees and clique trees, since these results are needed in the next section to study the relationships between ccts and scts. The reader may find it helpful to skip this section in a first reading of the paper.

Throughout this section, 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. 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, $hadj(w_0) = w_1 \cup hadj(w_1) \cdots = \{w_1, \ldots, w_i\} \cup hadj(w_i)$. 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 $\overline{new}(P) = \{w : w \in new(P),\ and\ w \geq f\}$. 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 the parent clique in a clique tree. The former path concatenated with the edge $(e, f)$ and the path $\overline{new}(P)$ constitutes the desired etree 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 $j = 1, \ldots, i$.

Now $f_i \in new(C_{i-1})$ 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 $w \geq f_1 \equiv first\_anc(C_1)$. 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 $v-w$ 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 $f_j \equiv first\_anc(C_j)$. 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 2 to vertex 6 in the etree in Fig. 1. The corresponding clique path in the first clique tree in Fig. 3 is $K(2)$, $K(3)$, $K(5)$; further, this etree path is a subpath of $new(K(2)), \overline{new}(K(3)), \overline{new}(K(5))$.

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 [16] (see the properties of clique trees described in § 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', $\mathcal{M} = \{M_1, \ldots, M_i\}$, such that for each clique $M_j$ where $1 \leq j \leq i$, the following properties hold: (i). $v$ belongs to $M_j$ and to all cliques on the clique path from $M_j$ to $K$; (ii). $v$ does not belong to any descendant of $M_j$.

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 $\mathcal{M}$ 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 [22] characterizes the structure of the $v$-th row of $L$ as a pruned subtree $F(v)$ of the etree rooted at $v$. This result now follows easily from the previous Theorem since the cliques in $\mathcal{D}$ together with $K$ form a subtree of the clique tree rooted at $K$. The proof, which we omit, provides an intuitive understanding of why the row structure is a pruned subtree of the etree.

The reader can verify that for the example we have considered, $ladj(6) = \{2, 3, 4, 5\}$; the row subtree is the pruned subtree of the etree rooted at 6 with leaves 2, 3, and 5; and that the cliques containing the vertex 6 form subtrees of both the clique trees we have obtained for this graph.

**5. Relationships between compact and skeleton clique trees.** In this section, we characterize the situations when the skeleton and compact cliques differ from each other, and prove that the cct is smaller in size than the sct. It may be helpful to omit the proofs in a first reading.

Let $K^S$, $K^C$ denote the skeleton clique and compact clique, respectively, of a maximal clique $K$ with respect to a clique tree $T$. 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 $F(w)$ (the pruned subtree of the etree rooted at $w$). We will make use of this fact several times in this section.

Our first result concerns a leaf clique.

LEMMA 5.1. *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 $v \cup S(v)$.*

*Proof.* Since $v$ is the representative vertex of $K$, $K = v \cup hadj(v)$, and because $K$ 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 $K^S = v \cup S(v)$. 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 $K^S = v \cup S(v)$ is a subset of the vertices in $v \cup hadj(v)$, 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. Define $low(K)$ to be the lowest-numbered vertex $v$ belonging to both $K$ and some child clique of $K$. It should be clear that $low(K)$ is equal to $\min_{C \in \mathcal{C}(K)} first\_anc(C)$, where $\mathcal{C}(K)$ is the set of child cliques of $K$. Using $low(K)$, 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 5.2. *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 [16] (see the discussion about the properties of clique trees in § 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 because $w_0$ is the lowest-numbered vertex in $K$, 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 $w_0 \cup S(w_0) \supseteq K$. Since $w_0$ is a leaf of the etree, the skeleton clique $K^S = w_0 \cup S(w_0)$, and hence $K^S \supseteq K$. As in

Lemma 5.1, since $S(w_0)$ is a subset of the vertices in $hadj(w_0)$, we have $K^S = w_0 \cup S(w_0) = K$. ∎

Since $K^C = K \setminus \cup_{C \in \mathcal{C}(K)} anc(C)$, for a clique in $\mathcal{I}_1$ the difference between $K^S$ and $K^C$ is the union of the ancestor sets of the child cliques of $K$. The first clique tree in Fig. 3 illustrates this case. Note that the clique $K = K(5) = \{5, 6, 7\}$ has $low(K) = 6$, which is greater than its representative vertex 5; hence the skeleton clique is the clique itself, while the compact clique $K^C = \{5\}$.

We now consider interior cliques in the group $\mathcal{I}_2$. 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 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, $low(K) = w_0$. If the set $\mathcal{C}_2$ is empty, then $E(K) = \emptyset$, and $K^C = K^S$.

Both clique trees in Fig. 3 illustrate this case. Consider the clique $K = K(3) = \{3, 4, 6, 7\}$ in the first clique tree. The child clique $K(1)$ belongs to the set $\mathcal{C}_1$, while the clique $K(2)$ 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 6; hence $E(K) = \{6\}$. The reader can verify that for $K = K(3)$ in the second clique tree, we also have $E(K) = \{6\}$. Recall that the skeleton clique $K^S = \{6\}$, and that the compact clique $K^C = \emptyset$.

LEMMA 5.3. *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 5.4. *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 $l \neq w_0$, and let $l$ belong to the *new* set of some clique $C_i$. 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, and let $l$ belong to the *new* set of a clique $C_i$. 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 $\mathit{first\_anc}(C_1) \leq w_0$. But $w_0 = low(K) = \min_{C \in \mathcal{C}(K)} \mathit{first\_anc}(C) \leq \mathit{first\_anc}(C_1)$, and thus we have $w_0 = \mathit{first\_anc}(C_1)$. 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 $anc(C_1)$. 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 $K = w_0 \cup hadj(w_0)$. Let $w_j$ be any vertex of $K$ that belongs to $S(w_0)$. 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 some clique $C \in \mathcal{C}_1$, 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 $new(J)$, where $J$ is an ancestor clique of $K$. Hence there is a clique path from $C$ to $J$ such that $u \in new(C)$ and $w_j \in new(J)$. 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 clique trees and skeleton clique trees by means of the above results.

THEOREM 5.5.

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

*Proof.* Immediate by summing over all the vertices of the cct and sct, using Lemmas 5.1, 5.2, and 5.4. ■

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 5.6.

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

*Proof.* The first inequality follows from Theorem 5.5. 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 $K^S = v \cup S(v)$. For every other clique $K$, $K^S = S(w)$, where $w$ is the representative vertex of $K$. Thus the size of the sct is equal to 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 $T$. ■

**6. Some special cases.** In this section we consider two examples: the grid graph and the arrowhead graph, and describe a result that shows that a compact clique and its corresponding skeleton clique are identical in a situation that occurs frequently when the minimum degree or nested dissection ordering algorithms are used for ordering sparse matrices.

**Grid graphs.** Liu [18] shows that the skeleton graph of a $k \times k$ regular nine-point grid graph ordered by the nested dissection ordering described by George [11] 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 6.1. *Let $k = 2^m - 1$, where $m$ is a positive integer. Consider a $k \times k$ nine-point regular grid graph ordered by nested dissection ordering using the separators described by George [11]. This graph has a clique tree whose cct has 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.*

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 clique tree computed must have the property that the *first_anc* vertex of each clique is a *new* vertex in its parent—this example has clique trees which do not satisfy this property!) 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 most cliques $K$ belonging to class $\mathcal{I}_2$ have no child cliques belonging to $\mathcal{C}_2$, and hence the excess set of vertices $E(K)$ is empty. The reason why these ordering algorithms give rise to clique trees with the above properties is not well understood.

**A frequent situation**. We now describe a result which offers some additional justification of the fact the cct and the sct are experimentally close in size. It shows that in a situation that arises commonly when the above ordering algorithms are used, a compact clique and its corresponding skeleton clique are identical.

Let $C_1, \ldots, C_k$ be the children of a maximal clique $K$ in a clique tree $T$. For $i = 1, \ldots, k$, let $T_i$ denote the subtree of $T$ rooted at $C_i$, and define $V_i = \cup_{D \in T_i} new(D)$, i.e., the set of all *new* vertices in the cliques belonging to $T_i$. Let $V_0 = new(K)$, and let $G'$ denote the subgraph of the chordal graph $G$ induced by the vertices in $V' = \cup_{j=0}^{k} V_j$.

Suppose $v \in V_i$, and $w \in V_j$, with $i, j = 1, \ldots, k$, and $i \neq j$. We claim that $v$ and $w$ cannot be adjacent in $G$, and hence in $G'$. Without loss of generality, let $v < w$. If $v$ is adjacent to $w$, then since $v \in new(D)$ for some clique $D$ in $T_i$, we have $hadj(v) \subset D$, and thus $w \in D$. But then $w$ must belong to some ancestor clique of $D$ as a *new* vertex, and this contradicts the fact that $w$ belongs as a *new* vertex to some clique in $T_j$.

From the claim, we have that $V_0$ is a separator of $G'$, and the removal of $V_0$ from $G'$ creates connected components with vertex sets $V_1, \ldots, V_k$. When nested dissection or Multiple Minimum Degree ordering algorithms are used, quite often it turns out that $V_0$ is a *minimal separator* separating $V_1, \ldots, V_k$ in $G'$; i.e., no proper subset of $V_0$ is a separator of $G'$ separating the above sets. For instance, when the regular nine-point grid graph is ordered with the separators described by George [11], all cliques except the root clique satisfy this property. The reason why filled graphs from the commonly used ordering algorithms have this property is not well understood yet.

The next result, which was kindly suggested to us by Barry Peyton, shows that in this common situation, the compact clique $K^C$ is identical to the skeleton clique $K^S$.

THEOREM 6.2. *Let $C_1, \ldots, C_k$ be the child cliques of a maximal clique $K$ in a clique tree $T$ of a chordal graph $G$. With the notation as in the above paragraphs, if $new(K) = V_0$ is a minimal separator of $G'$ which separates it into connected components with vertex sets $V_1, \ldots, V_k$, then $K^C = K^S$.*

*Proof.* Suppose a vertex $v \in V_0$ is adjacent to some vertex $w \in V_i$, for $1 \leq i \leq k$. Let $w \in new(D)$, where $D$ is some clique belonging to $T_i$. Since $w$ is a *new* vertex in $D$, we have

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

| key | $n$ | $\|A\|$ | $\|L\|$ | ncsubs | $\|\text{CT}\|$ | $\|\text{CCT}\|$ | $\|\text{SCT}\|$ | $\|\text{CCT}\|/n$ |
|---|---|---|---|---|---|---|---|---|
| BCSPWR10 | 5.3 | 8.3 | 23 | 17 | 22 | 9.2 | 9.3 | 1.7 |
| GRID 80 9 | 6.4 | 25 | 183 | 44 | 48 | 14 | 14 | 2.2 |
| NASA2910 | 2.9 | 86 | 201 | 25 | 25 | 8.4 | 8.4 | 2.9 |
| BCSSTK16 | 4.9 | 140 | 740 | 50 | 51 | 12 | 12 | 2.4 |
| BCSSTK29 | 14 | 300 | 1,700 | 139 | 140 | 32 | 32 | 2.3 |
| BCSSTK31 | 35.6 | 570 | 5,300 | 328 | 340 | 79 | 79 | 2.2 |
| BCSSTK32 | 44.6 | 990 | 5,200 | 372 | 380 | 96 | 96 | 2.2 |
| NASARB | 54.9 | 1,300 | 11,900 | 590 | 600 | 141 | 142 | 2.6 |

$hadj(w) \subset D$, and thus $v \in anc(D)$. Further, since $v \in new(K)$, by a property of clique trees, $v$ belongs to all cliques on the clique path from $D$ to $K$. In particular, $v \in anc(C_i)$.

Now since $V_0$ is a minimal separator of $G'$, each vertex $v \in V_0$ is adjacent to some vertex in $V_i$, for every $i$ with $1 \leq i \leq k$. Then from the above paragraph, each vertex $v \in V_0$ belongs to every one of the sets $anc(C_i)$, for $i = 1, \ldots, k$. Thus $V_0 \subset anc(C_i)$, for every $i$. It follows that $low(K) = rep(K)$, and hence $K$ belongs to the set of interior cliques $\mathcal{I}_2$. Further, all the child cliques $C_1, \ldots, C_k$ have $first\_anc(C_j) = rep(K)$, and thus belong to the set $\mathcal{C}_1$. Hence the set $\mathcal{C}_2$ is empty. Then by its definition, the excess set of vertices $E(K)$ is empty and $K^C = K^S$. ∎

**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 [7]. The results we report were obtained by ordering the matrices using Liu's Multiple Minimum Degree (MMD) ordering [17]. All programs were written in C, and the computations were performed on one processor of a Cray-2 running Unicos 5.1, using version 5.0.2 of the C compiler.

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 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 $A$. The column 'ncsubs' shows the number of row subscripts in Sherman's compressed column storage scheme for $L$.

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

TABLE 2
*Times for computing the cct (in units of $10^{-2}$ sec on a CRAY-2).*

| key | MMD | $A \to L$ | $L \to$ CT | CT $\to$ CCT |
|---|---|---|---|---|
| BCSPWR10 | 47 | 9 | 2 | 7 |
| GRID 80 9 | 53 | 18 | 2 | 11 |
| NASA2910 | 94 | 59 | 1 | 5 |
| BCSSTK16 | 169 | 103 | 1 | 11 |
| BCSSTK29 | 786 | 191 | 3 | 30 |
| BCSSTK31 | 1,511 | 323 | 9 | 71 |
| BCSSTK32 | 1,528 | 620 | 11 | 78 |
| NASARB | 2,279 | 886 | 13 | 122 |

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 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. The sizes of the cct and the sct are almost the same for all these problems. We offered some justification for this in § 6.

In Tables 2 and 3, 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 times have been scaled for convenience; the unit of time is $10^{-2}$ seconds on a Cray-2. For comparison purposes, in the first of these tables, we report the times taken to order the matrices by the MMD algorithm. The computation of the cct involves three additional steps: first, a symbolic factorization step to compute $L$; next, the computation of the clique tree from $L$ using the clique tree algorithm described in [16]; and finally, the computation of the cct from the clique tree. Note that the symbolic factorization step dominates the other two steps. The symbolic factorization we used was a translation into C of the `symbfct` routine in Sparspak which uses a quadratic-time insertion-sort to sort the row subscripts of the columns during the factorization. For the large problems reported here, the use of a more efficient ($n \log n$ time, where $n$ is number of elements to be sorted) sorting routine would substantially decrease the symbolic factorization time. This would enable the cct to be computed faster. Unfortunately, we have not been able to obtain timing results on a Cray-2 after making this change.

The sct is obtained by computing the elimination tree, the higher degrees, and the skeleton matrix. First the row structures of the skeleton matrix are computed using an

TABLE 3

*Detailed timing results for computing the sct and for computing the clique tree from the sct (in units of $10^{-2}$ seconds on a Cray-2).*

| key | etree | hdegree | skeleton matrix | | sct alg | total | SCT $\rightarrow$ CT |
|---|---|---|---|---|---|---|---|
| | | | rows | cols | | $A \rightarrow$ SCT | |
| BCSPWR10 | 3 | 3 | 11 | 3 | 4 | 24 | 11 |
| GRID 80 9 | 9 | 19 | 19 | 4 | 5 | 56 | 11 |
| NASA2910 | 27 | 21 | 76 | 2 | 2 | 128 | 4 |
| BCSSTK16 | 45 | 73 | 105 | 3 | 4 | 230 | 7 |
| BCSSTK29 | 95 | 168 | 205 | 9 | 11 | 488 | 23 |
| BCSSTK31 | 182 | 524 | 385 | 22 | 27 | 1,140 | 55 |
| BCSSTK32 | 311 | 520 | 683 | 28 | 33 | 1,575 | 59 |
| NASARB | 413 | 1,194 | 888 | 39 | 43 | 2,577 | 85 |

algorithm described by Liu [18], and from these, the higher adjacency lists $S(v)$ are obtained by a linear-time bucket sort. Then the Skeleton Clique Tree Algorithm is used to compute the sct. The time required by each of these steps is shown in the second through sixth columns of Table 3, and the total time required to compute the sct is shown in the seventh column of the table. (The column 'rows' shows the time to compute the row structures of the skeleton matrix, and 'cols' shows the time for the bucket sort to obtain the column structures of the skeleton matrix.) Note that the computation of the sct requires more than twice the time required to compute the cct for these problems. 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 3 shows the time required to compute the clique tree from the sct. (Computing the clique tree from the cct takes similar amounts of time.) These times are substantially smaller than the symbolic factorization times (shown in Table 2), but this is caused primarily by the inefficient insertion sort in the symbolic factorization routine. We verified on a Sun 4/260 that the time taken by solely the merging operations in computing the clique tree from the sct was faster than the merging required in symbolic factorization, by computing the times taken by these parts of the code. For instance, on BCSSTK29, the former was faster by about thirty percent over the latter.

**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.** We describe some potential applications of the cct and sct data structures now. We are currently implementing some of these algorithms which make use of the cct and the sct, and these results will be described elsewhere.

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.

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 [23] is used in modern sparse matrix software [6, 12]. 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 [12] 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 [22] 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 [18] 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 [13] have extended this work to static storage implementations of Gaussian elimination and orthogonal factorization.

Bank and Smith [4] 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. Bank and Rose [3] have recently described a solution of this problem for grid graphs.

Eisenstat, Schultz, and Sherman [8] 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. 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. This makes the computation of the clique tree from the cct faster than symbolic factorization.

**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 [13].

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, clique trees with vastly different structures could be computed from the same PEO, and since a cct is defined with respect to a clique tree, the size of a cct depends on the clique tree chosen. Indeed, there are worst-case examples of chordal graphs such that it is possible to compute two ccts 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. We have offered a justification of this phenomenon when the *new* set of a maximal clique turns out to be a minimal separator in an induced subgraph. It would be worthwhile to understand why most cliques have this property when common ordering algorithms are used.

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.

**Acknowledgements**. We thank the referees, John Lewis, and Barry Peyton for helpful suggestions, and Phuong Vu for providing the opportunity for Chunguang Sun to visit Cray Research last summer. Thanks also to Chao Wu Yang at Cray for running some experiments during the revision of this paper.

## 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 Division, Boeing Computer Services, Seattle, WA, 1987.

[2] C. Ashcraft, R. Grimes, J. Lewis, B. Peyton, and H. Simon, *Progress in sparse matrix methods for large linear systems on vector supercomputers*, Int. J. Supercomputer Appl., 1 (1987), pp. 10–30.

[3] R. E. Bank and D. J. Rose, *On the complexity of sparse Gaussian elimination via bordering*, SIAM J. Sci. Stat. Comput., 11 (1990), pp. 145–160.

[4] R. E. Bank and R. K. Smith, *General sparse elimination requires no permanent integer storage*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 574–584.

[5] J. R. S. Blair, R. E. England, and M. G. Thomason, *Cliques and their separators in triangulated graphs*, Tech. Report CS73-88, Computer Science, University of Tennessee, Knoxville, Aug 1988.

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

[7] I. S. Duff, R. Grimes, and J. Lewis, *Sparse matrix test problems*, ACM Trans. on Math. Software, 15 (1989), pp. 1–14.

[8] S. Eisenstat, M. Schultz, and A. Sherman, *Software for sparse Gaussian elimination with limited core storage*, in Sparse Matrix Proceedings, I. Duff and G. Stewart, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1978, pp. 135–153.

[9] D. R. Fulkerson and O. A. Gross, *Incidence matrices and interval graphs*, Pacific J. Math., 15 (1965), pp. 835–855.

[10] F. Gavril, *The intersection graphs of subtrees are exactly the chordal graphs*, J. Comb. Theory, Ser. B, 16 (1974), pp. 47–56.

[11] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 15 (1978), pp. 1053–1069.

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

[13] ——, *Compact structural representation of sparse Cholesky, QR and LU factors*, in Computer Methods in Applied Sciences and Engineering VII, R. Glowinski and J.-L. Lions, eds., Elsevier Publishers B. V. (North Holland), 1986, pp. 93–106.

[14] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[15] J. Lewis, B. W. Peyton, and A. Pothen, *Supernodes and clique trees in sparse matrix factorizations.* work in preparation, 1989.

[16] J. G. Lewis, B. W. Peyton, and A. Pothen, *A fast algorithm for reordering sparse matrices for parallel factorization*, SIAM J. Sci. Stat. Comput., 6 (1989), pp. 1146–1173.

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

[18] ——, *A compact row storage scheme for Cholesky factors using elimination trees*, ACM Trans. on Math. Software, 12 (1986), pp. 127–148.

[19] ——, *The role of elimination trees in sparse factorization*, Tech. Report 87-12, Computer Science, York University, North York, Ontario, Canada, 1987.

[20] B. W. Peyton, *Some Applications of Clique Trees to the Solution of Sparse Linear Systems*, PhD thesis, Clemson University, Clemson, SC, 1986.

[21] A. Pothen, *Simplicial cliques, shortest elimination trees, and supernodes in sparse Cholesky factorization*, Tech. Report CS-88-13, Computer Science, The Pennsylvania State University, University Park, PA, April 1988.

[22] R. Schreiber, *A new implementation of sparse Gaussian elimination*, ACM Trans. on Math. Software, 8 (1982), pp. 256–276.

[23] A. H. Sherman, *On the efficient solution of sparse linear and nonlinear equations*, PhD thesis, Department of Computer Science, Yale University, 1975.

[24] R. E. 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.

[25] J. R. Walter, *Representations of chordal graphs as subtrees of a tree*, J. Graph Theory, 2 (1978), pp. 265–267.

[26] E. Zmijewski and J. R. Gilbert, *A parallel algorithm for sparse symbolic Cholesky factorization on a multiprocessor*, Parallel Computing, 7 (1988), pp. 199–210.