ORNL/TM--12203

DE93 012508

Engineering Physics and Mathematics Division

# AN INTRODUCTION TO CHORDAL GRAPHS AND CLIQUE TREES

Jean R. S. Blair †
Barry W. Peyton ‡

† Department of Computer Science
University of Tennessee
Knoxville, TN 37996-1301

‡ Mathematical Sciences Section
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

Date Published: November 1992

Research was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy.

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400

## Contents

# AN INTRODUCTION TO CHORDAL GRAPHS AND CLIQUE TREES

Jean R. S. Blair
Barry W. Peyton

**Abstract**

Clique trees and chordal graphs have carved out a niche for themselves in recent work on sparse matrix algorithms, due primarily to research questions associated with advanced computer architectures. This paper is a unified and elementary introduction to the standard characterizations of chordal graphs and clique trees. The pace is leisurely, as detailed proofs of all results are included. We also briefly discuss applications of chordal graphs and clique trees in sparse matrix computations.

## 1. Introduction

It is well known that *chordal graphs* model the sparsity structure of the Cholesky factor of a sparse positive definite matrix [39]. Of the many ways to represent a chordal graph, a particularly useful and compact representation is provided by *clique trees* [24,45].[1] Until recently, *explicit* use of the properties of chordal graphs or clique trees in sparse matrix computations was rarely needed. For example, chordal graphs are mentioned in a single exercise in George and Liu [16]. However, chordal graphs and clique trees have found a niche in more recent work in this area, primarily due to various research questions associated with advanced computer architectures. For instance, the multifrontal method [8], which was developed to obtain good performance on vector supercomputers, can be expressed very succinctly in terms of a clique tree representation of the underlying chordal graph [34,37].

This paper is intended as an update to the graph theoretical results presented and proved in Rose [39], which predated the introduction of clique trees. Our goal is to provide a unified introduction to chordal graphs and clique trees for those interested in sparse matrix computations, though we hope it will be of use to those in other application areas in which these graphs play a major role. We have striven to write a primer, not a survey article: we present a limited number of well known results of fundamental importance, and prove all the results in the paper. The pacing is intended to be leisurely, and the organization is intended to enable the reader to read selected topics of interest in detail.

The paper is organized as follows. Section 2 contains the standard well known characterizations of chordal graphs and presents the maximum cardinality search algorithm for computing a *perfect elimination ordering.* Section 3 presents several characterizations of the clique trees of a chordal graph, including a *maximum spanning tree* property that is probably not as widely known as the others are. Section 4 ties together certain concepts and results from the previous two sections: it identifies the *minimal vertex separators* in a chordal graph with edges in any one of its clique trees, and it also shows that the maximum cardinality search algorithm is just Prim's algorithm in disguise. Finally, Section 5 briefly discusses recent applications of chordal graphs and clique trees to specific questions arising in sparse matrix computations.

## 2. Chordal graphs

An undirected graph is *chordal* (*triangulated, rigid circuit*) if every cycle of length greater than three has a *chord*: namely, an edge connecting two nonconsecutive vertices on the cycle. After introducing graph notation and terminology in Section 2.1, we present two standard characterizations of chordal graphs in Sections 2.2 and 2.3.

[1] All technical terms used in this section are defined later in the paper.

The latter of these two sections shows that chordal graphs are characterized by possession of a *perfect elimination ordering* of the vertices. The *maximum cardinality search* algorithm is a linear-time procedure for generating a perfect elimination ordering. Section 2.4 describes this algorithm and proves it correct. The necessary definitions and references for each of these results are given in the appropriate subsection.

### 2.1. Graph terminology

We assume familiarity with elementary concepts and definitions from graph theory, such as tree, edge, undirected graph, connected component, etc. Golumbic [20] provides a good review of this material. Here we introduce some of the graph notation and terminology that will be used throughout the paper. Other concepts from graph theory will be introduced as needed in later sections of the paper.

We let $G = (V, E)$ denote an *undirected graph* with *vertex set* $V$ and *edge set* $E$. The number of vertices is denoted by $n = |V|$ and the number of edges by $e = |E|$. For any vertex set $S \subseteq V$, consider the edge set $E(S) \subseteq E$ given by

$$E(S) := \{(u, v) \in E \mid u, v \in S\}.$$

We let $G(S)$ denote the *subgraph of $G$ induced by $S$*, namely the subgraph $(S, E(S))$. At times it will be convenient to consider the induced subgraph of $G$ obtained by removing a set of vertices $S \subseteq V$ from the graph; hence we define $G \setminus S$ by

$$G \setminus S := G(V - S).$$

Two vertices $u, v \in V$ are said to be *adjacent* if $(u, v) \in E$. Also, the edge $(u, v) \in E$ is said to be *incident with* both vertices $u$ and $v$. The set of vertices adjacent to $v$ in $G$ is denoted by $adj_G(v)$. Similarly, the set of vertices adjacent to $S \subseteq V$ in $G$ is given by

$$adj_G(S) := \{v \in V \mid v \notin S \text{ and } (u, v) \in E \text{ for some vertex } u \in S\}.$$

(The subscript $G$ often will be suppressed when the graph is known by context.) An induced subgraph $G(S)$ is *complete* if the vertices in $S$ are pairwise adjacent in $G$. In this case we also say that $S$ *is complete in* $G$.

We let $[v_0, v_1, \ldots, v_k]$ denote a *simple path* of length $k$ from $v_0$ to $v_k$ in $G$, i.e., $v_i \neq v_j$ for $i \neq j$ and $(v_i, v_{i+1}) \in E$ for $0 \leq i \leq k - 1$. Similarly, $[v_0, v_1, \ldots, v_k, v_0]$ denotes a *simple cycle* of length $k + 1$ in $G$. Finally, a *chord* of a path (cycle) is any edge joining two nonconsecutive vertices of the path (cycle).

**Definition 1.** *An undirected graph $G = (V, E)$ is chordal (triangulated, rigid circuit) if every cycle of length greater than three has a chord.*

Clearly, any induced subgraph of a chordal graph is also chordal, a fact that is

useful in several of the proofs that follow.

### 2.2. Minimal vertex separators

A subset $S \subset V$ is a *separator* of $G$ if two vertices in the same connected component of $G$ are in two distinct connected components of $G \setminus S$. If $a$ and $b$ are two vertices separated by $S$ then $S$ is said to be an *ab-separator*. The set $S$ is a *minimal separator* of $G$ if $S$ is a separator and no proper subset of $S$ separates the graph; likewise $S$ is a *minimal ab-separator* if $S$ is an $ab$-separator and no proper subset of $S$ separates $a$ and $b$ into distinct connected components. When the pair of vertices remains unspecified, we refer to $S$ as a *minimal vertex separator*. It does not necessarily follow that a minimal vertex separator is also a minimal separator of the graph. For instance, in Figure 2.1 the set $S = \{b, e\}$ is a minimal $dc$-separator; nevertheless, $S$ is not a minimal separator of $G$ since $\{e\} \subset S$ is also a separator of $G$. Minimal vertex separators are used to

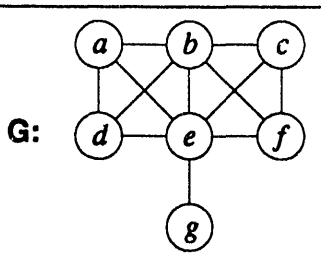


Figure 2.1: Minimal $dc$-separator $\{b, e\}$ is not a minimal separator of $G$.

characterize chordal graphs in Theorem 2.1, which is due to Dirac [6]. The proof is taken from Peyton [34], which, in turn, closely follows the proof given by Golumbic [20].

**Theorem 2.1 (Dirac [6]).** *A graph $G$ is chordal if and only if every minimal vertex separator of $G$ is complete in $G$.*

**Proof:** Assume every minimal vertex separator of $G$ is complete in $G$, and let $\mu = [v_0, \ldots, v_k, v_0]$ be any cycle of length greater than three in $G$ (i.e., $k \geq 3$). If $(v_0, v_2) \in E$, then $\mu$ has a chord. If not, then there exists a $v_0v_2$-separator $S$ (e.g., $S = V - \{v_0, v_2\}$); furthermore, any such separator must contain $v_1$ and $v_i$ for some $i$, $3 \leq i \leq k$. Choose $S$ to be a minimal $v_0v_2$-separator so that $S$, by assumption, is complete in $G$. It follows that $(v_1, v_i)$ is a chord of $\mu$, which proves the "if" part of the result.

Now assume $G$ is chordal and let $S$ be a minimal $ab$-separator of $G$. Let $G(A)$ and $G(B)$ be the connected components of $G \setminus S$ containing $a$ and $b$, respectively. It suffices to show that for any two distinct vertices in $S$, say $x$ and $y$, we have $(x, y) \in E$. Since $S$ is minimal, each vertex $v \in S$ is adjacent to some vertex in $A$ and some vertex in $B$; otherwise, $S - \{v\}$ would be an $ab$-separator contrary to the minimality of $S$. Thus, there exist paths $\mu = [x, a_1, \ldots, a_r, y]$ and $\nu = [y, b_1, \ldots, b_t, x]$ where each $a_i \in A$ and each $b_i \in B$ (see Figure 2.2). Further, choose $\mu$ and $\nu$ so that they are

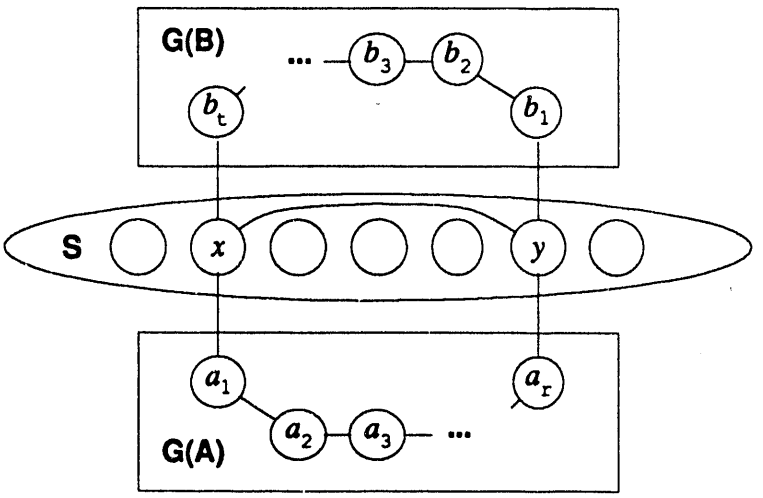


Figure 2.2: Cycle in proof of Theorem 2.1 that induces chord $(x, y)$.

of the smallest possible length greater than one, and combine them to form the cycle $\sigma = [x, a_1, \ldots, a_r, y, b_1, \ldots, b_t, x]$. Since $G$ is chordal and $\sigma$ is a cycle of length greater than three, $\sigma$ must have a chord. Any chord of $\sigma$ incident with $a_i$, $1 \leq i \leq r$, would either join $a_i$ to another vertex in $\mu$ contrary to the minimality of $r$, or would join $a_i$ to a vertex in $B$, which is impossible because $S$ separates $A$ from $B$ in $G$. Consequently, no chord of $\sigma$ is incident with a vertex $a_i$, $1 \leq i \leq r$, and by the same argument no chord of the cycle is incident with a vertex $b_j$, $1 \leq j \leq t$. It follows that the only possible chord is $(x, y)$. ∎

**Remark** In reality, $r = t = 1$, otherwise $[x, a_1, \ldots, a_r, y, x]$ or $[y, b_1, \ldots, b_t, x, y]$ is a chordless cycle of length greater than three.

### 2.3. Perfect elimination orderings

We need the following terminology before we can state and prove the main result in this section. An *ordering* $\alpha$ of $G$ is a bijection $\alpha : V \rightarrow \{1, 2, \ldots, n\}$. Often it will be convenient to denote an ordering by using it to index the vertex set, so that $\alpha(v_i) = i$ for $1 \leq i \leq n$ where $i$ will be referred to as the *label* of $v_i$. Let $v_1, v_2, \ldots, v_n$ be an ordering of $V$. For $1 \leq i \leq n$, we define $\mathcal{L}_i$ to be the set of vertices with labels greater than $i - 1$:

$$\mathcal{L}_i := \{v_i, v_{i+1}, \ldots, v_n\}.$$

The *monotone adjacency set* of $v_i$, denoted $madj_G(v_i)$, is given by

$$madj_G(v_i) := \{v_j \in adj(v_i) \mid j > i\} = adj_G(v_i) \cap \mathcal{L}_{i+1}.$$

Again, the subscript $G$ often will be suppressed where the graph is known by context.

A vertex $v$ is *simplicial* if $adj(v)$ induces a complete subgraph of $G$. The ordering $\alpha$ is a *perfect elimination ordering* (PEO) if for $1 \leq i \leq n$, the vertex $v_i$ is simplicial in the graph $G(\mathcal{L}_i)$. As shown below in Lemma 1, every nontrivial chordal graph has a simplicial vertex (actually, at least two). Theorem 2.2, which states that chordal graphs are characterized by the possession of a PEO, follows easily from Lemma 1. The proofs are again taken from Peyton [34], which, in turn, closely follow arguments found in Golumbic [20].

**Lemma 1 (Dirac [6]).** *Every chordal graph $G$ has a simplicial vertex. If $G$ is not complete, then it has two nonadjacent simplicial vertices.*

**Proof:** The lemma is trivial if $G$ is complete. For the case where $G$ is not complete we proceed by induction on the number of vertices $n$. Let $G$ be a chordal graph with $n \geq 2$ vertices, including two nonadjacent vertices $a$ and $b$. If $n = 2$, both vertices of the graph are simplicial since both are isolated (i.e., $adj(a) = adj(b) = \emptyset$). Suppose $n > 2$ and assume that the lemma holds for all such graphs with fewer than $n$ vertices. Since $a$ and $b$ are nonadjacent, there exists an $ab$-separator (e.g., the set $V - \{a, b\}$). Suppose $S$ is a minimal $ab$-separator of $G$, and let $G(A)$ and $G(B)$ be the connected components of $G \setminus S$ containing $a$ and $b$, respectively. The induced subgraph $G(A \cup S)$ is a chordal graph having fewer vertices than $G$; hence, by the induction hypothesis one of the following must hold: Either $G(A \cup S)$ is complete and every vertex of $A$ is a simplicial vertex of $G(A \cup S)$, or $G(A \cup S)$ has two nonadjacent simplicial vertices, one of which must be in $A$ since, by Theorem 2.1, $S$ is complete in $G$. Because $adj_G(A) \subseteq A \cup S$, every simplicial vertex of $G(A \cup S)$ in $A$ is also a simplicial vertex of $G$. By the same argument, $B$ also contains a simplicial vertex of $G$, thereby completing the proof. ∎

**Theorem 2.2 (Fulkerson and Gross [10]).** *A graph $G$ is chordal if and only if $G$ has a perfect elimination ordering.*

**Proof:** Suppose $G$ is chordal. We proceed by induction on the number of vertices $n$ to show the existence of a PEO of $G$. The case $n = 1$ is trivial. Suppose $n > 1$ and every chordal graph with fewer vertices has a PEO. By Lemma 1, $G$ has a simplicial vertex, say $v$. Now $G \setminus \{v\}$ is a chordal graph with fewer vertices than $G$; hence, by induction it has a PEO, say $\beta$. If $\alpha$ orders the vertex $v$ first, followed by the remaining vertices of $G$ in the order determined by $\beta$, then $\alpha$ is a PEO of $G$.

Conversely, suppose $G$ has a PEO, say $\alpha$, given by $v_1, v_2, \ldots, v_n$. We seek a chord of an arbitrary cycle $\mu$ in $G$ of length greater than three. Let $v_i$ be the vertex on $\mu$ whose label $i$ is smaller than that of any other vertex on $\mu$ Since $\alpha$ is a PEO, $madj(v_i)$ is complete; whence $\mu$ has at least one chord: namely, the edge joining the two neighboring vertices of $v_i$ in $\mu$. ∎

### 2.4. Maximum cardinality search

Rose, Tarjan, and Lueker [40] introduced the first linear-time algorithm for producing a PEO, known as the *lexicographic breadth first search* algorithm. In a set of unpublished lecture notes, Tarjan [43] introduced a simpler algorithm known as the *maximum cardinality search* (MCS) algorithm. Tarjan and Yannakakis [45] later described MCS algorithms for both chordal graphs and acyclic hypergraphs. The MCS algorithm for chordal graphs orders the vertices in reverse order beginning with an arbitrary vertex $v \in V$ for which it sets $\alpha(v) = n$. At each step the algorithm selects as the next vertex to label an unlabeled vertex adjacent to the largest number of labeled vertices, with ties broken arbitrarily. A high-level description of the algorithm is given in Figure 2.3. We refer the reader to Tarjan and Yannakakis [45] for details on how to implement the algorithm to run in $O(n + e)$ time.

---

$\mathcal{L}_{n+1} \leftarrow \emptyset$;
**for** $i \leftarrow n$ **to** 1 **step** $-1$ **do**
    Choose a vertex $v \in V - \mathcal{L}_{i+1}$ for which
        $|adj(v) \cap \mathcal{L}_{i+1}|$ is maximum;
    $\alpha(v) \leftarrow i$; [$v$ becomes $v_i$]
    $\mathcal{L}_i \leftarrow \mathcal{L}_{i+1} \cup \{v_i\}$;
**end for**

Figure 2.3: Maximum cardinality search (MCS).

---

The following lemma and theorem prove that the MCS algorithm produces a PEO. The lemma provides a useful characterization of the orderings of a chordal graph that are *not* perfect elimination orderings. Edelman, Jamison, and Shier [9,42] prove similar results while studying the notion of convexity in chordal graphs. Theorem 2.3 is then proved by showing that every ordering that is *not* a PEO is also *not* an MCS ordering. The proof is taken from Peyton [34]. Later in Section 4.2, we will provide a more intuitive view of how the MCS algorithm works: it can be viewed as a special implementation of Prim's algorithm applied to the *weighted clique intersection graph of* $G$ (defined in Section 3.4).

**Lemma 2.** *An ordering $\alpha$ of the vertices in a graph $G$ is not a perfect elimination ordering if and only if for some vertex $v$, there exists a chordless path of length greater than one from $v = \alpha^{-1}(i)$ to some vertex in $\mathcal{L}_{i+1}$ through vertices in $V - \mathcal{L}_i$.*

**Proof:** Suppose $\alpha$ is *not* a PEO. There exists then by Lemma 1 a vertex $u \in V$ for which $madj(u)$ is not complete in $G$; hence, there exist two vertices $v, w \in madj(u)$ joined by no edge in $E$. Without loss of generality assume that $i = \alpha(v) < \alpha(w)$.

Then $[v, u, w]$ is a chordless path of length two from $v = \alpha^{-1}(i)$ to $w \in \mathcal{L}_{i+1}$ through $u \in V - \mathcal{L}_i$.

Conversely, suppose there exists a chordless path $\mu = [u_0, u_1, \ldots, u_r]$ of length $r \geq 2$ from $u_0 = \alpha^{-1}(i)$ to $u_r \in \mathcal{L}_{i+1}$ through vertices $u_j \in V - \mathcal{L}_i$, $1 \leq j \leq r-1$. Let $u_k$, where $1 \leq k \leq r-1$, be the internal vertex in $\mu$ whose label $\alpha(u_k)$ is smaller than that of any other internal vertex in $\mu$. Then $madj(u_k)$ includes two nonadjacent vertices: namely, the two neighboring vertices of $u_k$ in $\mu$. It follows that $\alpha$ is not a PEO. ∎

**Theorem 2.3 (Tarjan [43], Tarjan and Yannakakis [45]).** *Every maximum cardinality search ordering of a chordal graph $G$ is a perfect elimination ordering.*

**Proof:** Let $\alpha$ be any ordering of a chordal graph $G$ that is *not* a PEO. We will show that the ordering $\alpha$ *cannot* be generated by the MCS algorithm.

By Lemma 2, for some vertex $u_0$ there exists a chordless path $\mu = [u_0, u_1, \ldots, u_{r-1}, u_r]$ of length $r \geq 2$ from $u_0 = \alpha^{-1}(i)$ to $u_r \in \mathcal{L}_{i+1}$ through vertices $u_j \in V - \mathcal{L}_i$, $1 \leq j \leq r-1$. (See Figure 2.4.) Choose $u_0$ so that the label $i = \alpha(u_0)$ is maximum among all the vertices of $G$ for which such a chordless path exists.

To show that $\alpha$ is not an MCS ordering it suffices to show that there exists some vertex $w \in V - \mathcal{L}_{i+1}$ for which $|adj(w) \cap \mathcal{L}_{i+1}|$ exceeds $|adj(u_0) \cap \mathcal{L}_{i+1}|$. We will show that the vertex $u_{r-1} \in \mu$ is indeed such a vertex. Note that $adj(u_0) \cap \mathcal{L}_{i+1}$ and $madj(u_0)$ are by definition identical, and thus it suffices to show that

$$madj(u_0) \subset adj(u_{r-1}) \cap \mathcal{L}_{i+1}. \tag{2.1}$$

For the trivial case $madj(u_0) = \emptyset$, the theorem holds since $u_{r-1}$ is adjacent to $u_r \in \mathcal{L}_{i+1}$. Assume instead that $madj(u_0) \neq \emptyset$, and choose a vertex $x \in madj(u_0)$. To see that $x$ is also adjacent to $u_{r-1}$, consider the path $\gamma = [x, u_0, \ldots, u_{r-1}, u_r]$ pictured in Figure 2.4. The maximality of $i$ implies that every path of length greater than one having the following two properties will have a chord: a) the endpoints of the path are both numbered greater than $i$, and b) the interior vertices are numbered less than the minimum of the endpoints. The path $\gamma$ satisfies these two properties and hence has a chord. Moreover, since $\mu = [u_0, u_1, \ldots, u_r]$ has no chords, every chord of $\gamma$ is incident with $x$. Let $u_k$ be the vertex in $\gamma$ adjacent to $x$ which has the largest subscript. If $k \neq r$ then $[x, u_k, \ldots, u_r]$ is a chordless path, again contrary to the maximality of $i$; hence $(x, u_r) \in E$.

It follows that $\sigma = [x, u_0, \ldots, u_{r-1}, u_r, x]$ is a cycle of length greater than three in $G$ (recall that $r \geq 2$). Since $G$ is chordal, $\sigma$ must have a chord, and, as argued above, any such chord must be incident with $x$. Let $u_t$ be the vertex in $\sigma$ with the highest subscript other than $r$, for which $(x, u_t) \in E$. If $t \neq r-1$, then $[x, u_t, \ldots, u_r, x]$ is a chordless cycle of length greater than 3, contrary to the chordality of $G$. In consequence, $(x, u_{r-1}) \in E$ for all $x \in madj(u_0)$. But $u_{r-1}$ is also adjacent to $u_r \in \mathcal{L}_{i+1} - madj(u_0)$, whence (2.1) holds, completing the proof. ∎

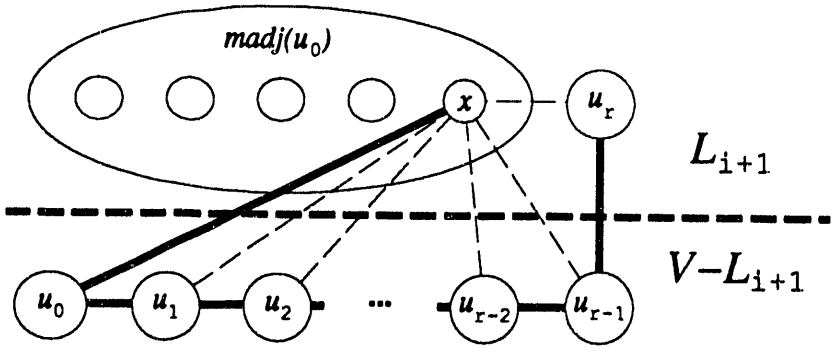


Figure 2.4: Illustration for the proof of Theorem 2.3. The dark solid edges exist by hypothesis; existence of the lighter broken edges is argued in the proof and the remark that follows it.

**Remark** In the preceding proof the argument leading to the inclusion of $(x, u_{r-1})$ in $E$ can be repeated for every edge $(x, u_j)$, $1 \leq j \leq r-2$. In consequence we have

$$madj(u_0) \subseteq adj(u_j) \cap \mathcal{L}_{i+1} \text{ for } 1 \leq j \leq r-2. \tag{2.2}$$

Statement (2.1) implies that if the MCS algorithm "tried" to generate $\alpha$, then as the vertex to be labeled with $i$ is chosen, the priority of $u_{r-1}$ would be greater than that of $u_0$. Similarly, (2.2) implies that the priority of each vertex $u_j$ $(1 \leq j \leq r-2)$ would be at least as great as that of $u_0$.

## 3. Characterizations of clique trees

Let $G = (V, E)$ be any graph. A *clique* of $G$ is any *maximal* set of vertices that is complete in $G$, and thus a clique is properly contained in no other clique. We will refer to a "submaximal clique" as complete in $G$, as we did in the previous section. Henceforth $\mathcal{K}_G = \{K_1, K_2, \ldots, K_m\}$ denotes the set containing the cliques of $G$, and $m$ will be the number of cliques.

The reader may verify that the graph in Figure 3.1 is a chordal graph with four cliques, each of size three. The graph in Figure 3.1 will be used throughout this section to illustrate results and key points. For convenience we shall refer to the vertices of this graph as $v_1, v_2, \ldots, v_7$; e.g., the vertex labeled "6" will be referred to as $v_6$. Note that the labeling of the vertices is a PEO of the graph.

For any chordal graph $G$ there exists a subset of the set of trees on $\mathcal{K}_G$ known as *clique trees*. Any one of these clique trees can be used to represent the graph, often in a very compact and efficient manner [24,46], as we shall see in Section 4. This section contains a unified and elementary presentation of several key properties of clique trees, each of which has been shown, somewhere in the literature, to characterize the set of

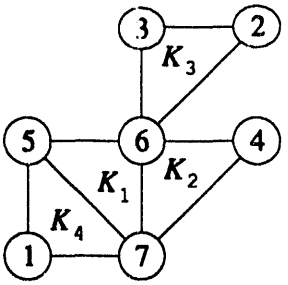


Figure 3.1: Chordal graph with seven vertices and four cliques.

clique trees associated with a chordal graph.

The notion of clique trees was introduced independently by Buneman [5], Gavril [12], and Walter [46]. The property we use to introduce and define clique trees in Section 3.1 is a simple variant of one of the key properties introduced in their work. We use this variant because, in our experience, it is more readily apprehended by those who are studying this material for the first time. Section 3.2 presents the short argument needed to show that the more recent variant is equivalent to the original.

Clique trees have found application in relational databases, where they can be viewed as a subclass of *acyclic hypergraphs*, which are heavily used in that area. Open problems in relational database theory motivated the pioneering work of Bernstein and Goodman [2], Beeri, Fagin, Maier, and Yannakakis [1], and Tarjan and Yannakakis [45]. Our two final characterizations of clique trees, presented in Sections 3.3 and 3.4, are based on results from these papers. Section 3.5 summarizes these results, and also illustrates these results in negative form using the example in Figure 3.1.

Throughout this section it will be convenient to assume that $G$ is connected. All the results can nevertheless be applied to a disconnected graph by applying them successively to each connected component; thus no loss of generality is incurred by the restriction. Note also that Sections 3.2, 3.3, and 3.4 can be read independently of one another, but any of these three subsections should be read only after reading Section 3.1. As in the previous section, needed definitions and specific references to the literature are given in the appropriate subsections.

### 3.1. Definition using the clique-intersection property

Assume that $G$ is a connected graph (not necessarily chordal), and consider its set of maximal cliques $\mathcal{K}_G$. In this section we consider the set of trees on $\mathcal{K}_G$ that satisfy the following *clique-intersection* property:

> For every pair of distinct cliques $K, K' \in \mathcal{K}_G$, the set $K \cap K'$ is contained in every clique on the path connecting $K$ and $K'$ in the tree.

As an example of a tree that satisfies the clique-intersection property, consider the tree shown in Figure 3.2, whose vertices are the cliques of the chordal graph in Figure 3.1. The reader may verify that this tree indeed satisfies the clique-intersection property:

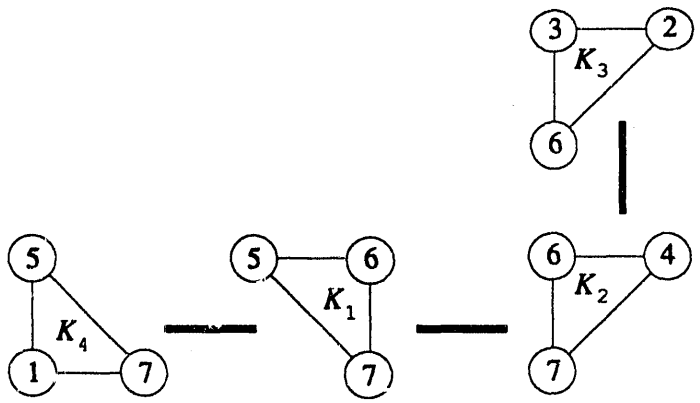


Figure 3.2: A tree on the cliques of the chordal graph in Figure 3.1, which satisfies the clique-intersection property.

for example, the set $K_4 \cap K_2 = \{v_7\}$ is contained in $K_1$, which is the only clique on the path from $K_4$ to $K_2$ in the tree. The reader may also verify that the only other tree on $\{K_1, K_2, K_3, K_4\}$ that satisfies the clique-intersection property is obtained from the tree in Figure 3.2 by replacing the edge $(K_3, K_2)$ with $(K_3, K_1)$.

We will show in Theorem 3.1 below that $G$ is chordal if and only if there exists a tree on $\mathcal{K}_G$ that satisfies the clique-intersection property. For any given chordal graph $G$, we shall let $\mathcal{T}_G^{ct}$ denote the nonempty set of trees $T = (\mathcal{K}_G, \mathcal{E}_T)$ that satisfy the clique intersection property, and we shall refer to any member of $\mathcal{T}_G^{ct}$ as a *clique tree* of the underlying chordal graph $G$. In Section 3.2, we prove the original version of this result, which was introduced independently by Buneman [5], Gavril [12], and Walter [46].

To prove the main result of this subsection, we require two more definitions and a simple lemma. A vertex $K$ in a tree $T$ is a *leaf* if it has precisely one neighbor in $T$ (i.e., $|adj_T(K)| = 1$). We let $\mathcal{K}_G(v) \subseteq \mathcal{K}_G$ denote the set of cliques containing the vertex $v$. The following simple characterization of simplicial vertices has been useful in various applications. This result has been used widely in the literature [7,19,23,24,45], and has been formally stated and proven in at least two places [23,24].

**Lemma 3.** *A vertex is simplicial if and only if it belongs to precisely one clique.*

**Proof:** Suppose a vertex $v$ belongs to two cliques $K, K' \in \mathcal{K}_G$. Maximality of the cliques implies the existence of two distinct nonadjacent vertices $u \in K - K'$ and $u' \in K' - K$. Since both $u$ and $u'$ are adjacent to $v$, it follows that $v$ is not simplicial.

Assume now that the vertex $v$ belongs to one and only one clique $K \in \mathcal{K}_G$. Note that $v$ is adjacent to a vertex $u \neq v$ if and only if there exists a clique of $G$ to which both $u$ and $v$ belong. Consequently $adj(v) = K - \{v\}$, whence $v$ is simplicial. ∎

The first part of the following proof closely resembles the argument given by Gavril [12] to prove a result that shall be presented in the next section. The second half was improvised for this paper, and resembles the first half in many of its features.

**Theorem 3.1.** *A connected graph $G$ is chordal if and only if there exists a tree $T = (\mathcal{K}_G, \mathcal{E}_T)$ for which the clique-intersection property holds.*

**Proof:** We proceed by induction on the number of vertices $n$ to show the "only if" part. The base step $n = 1$ is obvious. For the induction step, let $G$ be a chordal graph with $n \geq 2$ vertices and assume the result is true for all chordal graphs having fewer than $n$ vertices. By Lemma 1, $G$ has a simplicial vertex, say $v$. Let $K$ be the single clique of $G$ that contains $v$ (see Lemma 3), and consider the induced subgraph $G' = G \backslash \{v\}$. Since $G'$ is a chordal graph with $n-1$ vertices, by the induction hypothesis there exists a tree $T' = (\mathcal{K}_{G'}, \mathcal{E}_{T'})$ that satisfies the clique-intersection property.

To complete the proof of the "only if" part, there are two cases to consider. First, suppose $K' = K - \{v\}$ remains maximal in $G'$ (i.e., $K' \in \mathcal{K}_{G'}$). It is trivial to show that $\mathcal{K}_{G'} = \mathcal{K}_G \cup \{K'\} - \{K\}$, and we leave it for the reader to verify this. It follows that the only difference between the cliques of $G$ and $G'$ is the presence in $G$ of the simplicial vertex $v$ in $K$ and the absence of $v$ from the corresponding clique $K'$ of $G'$. In consequence, the intersection of any pair of cliques in $G$ is identical to the intersection of the corresponding pair in $G'$. Let $T$ be the tree on $\mathcal{K}_G$ obtained from $T'$ by replacing $K'$ with $K$. Since $T'$ has the clique-intersection property, it follows that $T$ has this property as well, thereby completing the argument for the first case.

Now, suppose $S' = K - \{v\}$ is not a maximal clique in $G'$ (i.e., $S' \notin \mathcal{K}_{G'}$). Since $n \geq 2$ and $G$ is connected, $v$ is not an isolated vertex, and we have

$$S' = K - \{v\} = adj(v) \neq \emptyset.$$

Since $S'$ is complete in $G'$, there exists a clique $P \in \mathcal{K}_{G'} = \mathcal{K}_G - \{K\}$ for which $S' \subset P$. (As before, we leave it for the reader to verify that $\mathcal{K}_{G'} = \mathcal{K}_G - \{K\}$.) Let $T$ be the tree on $\mathcal{K}_G$ obtained by adding the clique $K$ and the edge $(K, P)$ to $T'$. We now verify that $T$ satisfies the clique-intersection property. Because $T'$ satisfies the clique-intersection property, the set $K_1 \cap K_2$ is contained in every clique on the path from $K_1$ to $K_2$ in $T$ whenever neither $K_1$ nor $K_2$ is $K$. Consider now the set $K \cap K''$ where $K'' \in \mathcal{K}_G - \{K\} = \mathcal{K}_{G'}$. Since $K - \{v\} \subset P$ and $v$ belongs to no clique in $\mathcal{K}_G - \{K\}$, it follows that $K'' \cap K \subset P$. Because $T'$ satisfies the clique-intersection property, the set $K \cap K'' = P \cap K''$ is contained in every clique on the path from $K$ to $K''$ in $T$, and $T$ therefore satisfies the clique-intersection property as well.

To prove the "if" part, let $G = (V, E)$ be a graph and suppose there exists a tree $T = (\mathcal{K}_G, \mathcal{E}_T)$ that satisfies the clique-intersection property. Again we proceed by induction on $n$ to show that $G$ is chordal. The base step $n = 1$ is obvious. For the induction step, let $G$ be a graph with $n \geq 2$ vertices and assume the result is true for all graphs having fewer than $n$ vertices.

Let $K$ and $P$ be respectively a leaf of $T$ and its sole neighbor (i.e., "parent") in $T$. By maximality of the cliques there exists a vertex $v \in K - P$. The vertex $v$ moreover cannot belong to any clique $K' \in \mathcal{K}_G - \{K, P\}$, for were it otherwise the clique $P$, which is on the path from $K$ to $K'$ in $T$, would not contain the set $K \cap K'$. Consequently $v$ belongs to no other clique but $K$, whence by Lemma 3 it is a simplicial vertex of $G$.

Consider the reduced graph $G' = G \setminus \{v\}$ and let $K' = K - \{v\}$. If $K' \not\subset P$, then the "reduced" tree $T'$ for $G'$ is obtained simply by replacing $K$ with $K'$ in $T$; if $K' \subset P$, then $T'$ is obtained by removing from $T$ the vertex $K$ and the single edge $(K, P)$ incident with it in $T$. As before, in the first case, $\mathcal{K}_{G'} = \mathcal{K}_G \cup \{K'\} - \{K\}$; in the second case, $\mathcal{K}_{G'} = \mathcal{K}_G - \{K\}$. In either case, it is trivial to verify that the tree $T'$ satisfies the clique-intersection property. From the induction hypothesis it follows that $G'$ is chordal. Let $\beta$ be any PEO of $G'$. A PEO of $G$ can then be obtained by ordering $v$ first, followed by the remaining vertices of $G$ in the order determined by $\beta$. Thus by Theorem 2.2, $G$ is also chordal, giving us the result. ∎

### 3.2. The induced-subtree property

In this section we are concerned with the set of all trees on $\mathcal{K}_G$ that satisfy the *induced-subtree* property:

> For every vertex $v \in V$, the set $\mathcal{K}_G(v)$ induces a subtree of $T$.

We shall let $\mathcal{T}_G^{\text{ist}}$ denote the set of all trees on $\mathcal{K}_G$ that satisfy the induced-subtree property.

Consider again the clique tree in Figure 3.2. Observe that each of the sets $\mathcal{K}_G(v_3) = \{K_3\}$ and $\mathcal{K}_G(v_6) = \{K_1, K_2, K_3\}$ induces a subtree of this tree. The reader may verify that this tree satisfies the induced-subtree property. It is trivial to prove that the clique-intersection and induced-subtree properties are indeed equivalent.

**Theorem 3.2.** *For any connected graph $G$, we have $\mathcal{T}_G^{\text{ist}} = \mathcal{T}_G^{\text{ct}}$.*

**Proof:** To see that $\mathcal{T}_G^{\text{ct}} \subseteq \mathcal{T}_G^{\text{ist}}$, let $T_{\text{ct}} \in \mathcal{T}_G^{\text{ct}}$ and consider the set of cliques $\mathcal{K}_G(v)$ for some vertex $v \in V$. Choose two cliques $K, K' \in \mathcal{K}_G(v)$. Since the set $K \cap K'$ lies in every clique on the path joining $K$ and $K'$ in $T_{\text{ct}}$, it follows that the vertex $v \in K \cap K'$ also lies in each clique along this path. In consequence, the induced subgraph $T_{\text{ct}}(\mathcal{K}_G(v))$ is connected and hence a subtree of $G$. It follows that $T_{\text{ct}} \in \mathcal{T}_G^{\text{ist}}$, whence $\mathcal{T}_G^{\text{ct}} \subseteq \mathcal{T}_G^{\text{ct}}$, as desired.

To see that $\mathcal{T}_G^{\text{ist}} \subseteq \mathcal{T}_G^{\text{ct}}$, let $T_{\text{ist}} \in \mathcal{T}_G^{\text{ist}}$. Choose two cliques $K, K' \in \mathcal{K}_G$, and consider the set $K \cap K'$. For each vertex $v \in K \cap K'$, the set $\mathcal{K}_G(v)$ induces a subtree of $T_{\text{ist}}$ (i.e., a connected subgraph of $T_{\text{ist}}$); and thus the vertex $v$ lies in each clique along the path joining $K$ and $K'$ in $T_{\text{ist}}$. It follows that $T_{\text{ist}} \in \mathcal{T}_G^{\text{ct}}$, whence $\mathcal{T}_G^{\text{ist}} \subseteq \mathcal{T}_G^{\text{ct}}$, as desired. ∎

We thus have the following well known result from the literature.

**Theorem 3.3 (Buneman [5], Gavril [12], Walter [46]).** *A connected graph $G$ is chordal if and only if there exists a tree $T = (\mathcal{K}_G, \mathcal{E}_T)$ for which the induced-subtree property holds.*

**Proof:** The result follows immediately from Theorems 3.1 and 3.2. ∎

### 3.3. The running intersection property

A total ordering of the cliques in $\mathcal{K}_G$, say $K_1, K_2, \ldots, K_m$, has the *running intersection property* (RIP) if for each clique $K_j$, $2 \leq j \leq m$, there exists a clique $K_i$, $1 \leq i \leq j-1$, such that

$$K_j \cap (K_1 \cup K_2 \cup \cdots \cup K_{j-1}) \subset K_i. \tag{3.1}$$

For any RIP ordering of the cliques, we construct a tree $T_{\text{rip}}$ on $\mathcal{K}_G$ by making each clique $K_j$ adjacent to a "parent" clique $K_i$ identified by (3.1). (Since more than one clique $K_i$, $1 \leq i \leq j-1$, may satisfy (3.1), the parent may not be uniquely determined.) We let $\mathcal{T}_G^{\text{rip}}$ be the set containing every tree on $\mathcal{K}_G$ that can be constructed from an RIP ordering in this manner. We define a *reverse topological ordering* of any rooted tree as an ordering that numbers each parent *before* any of its children. Finally, note that any RIP ordering is a reverse topological ordering of a rooted tree constructed from the ordering in the manner specified above.

The ordering $K_1, K_2, K_3, K_4$ of the cliques shown in Figure 3.1 is an RIP ordering; a corresponding RIP-induced parent function is displayed in Figure 3.3. Note that the parent function specifies precisely the edges of the clique tree in Figure 3.2. Indeed, we can show that for any connected graph $G$, we have $\mathcal{T}_G^{\text{rip}} = \mathcal{T}_G^{\text{ct}}$.

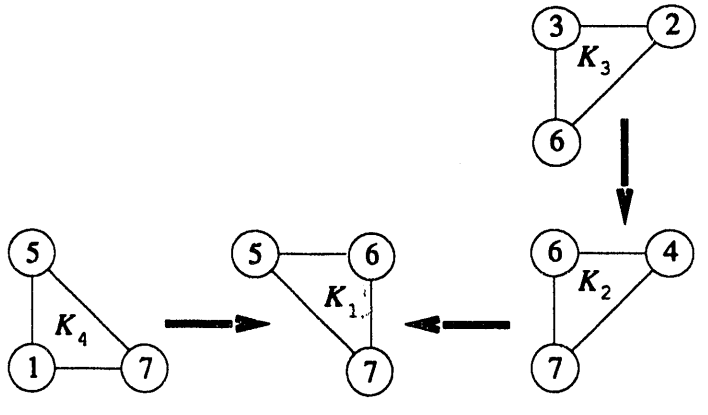


Figure 3.3: Clique tree in Figure 3.2 is an RIP tree. Arrows point from child to parent.

**Theorem 3.4 (Beeri, Fagin, Maier, Yannakakis [1]).** *For any connected graph $G$, we have $\mathcal{T}_G^{\text{rip}} = \mathcal{T}_G^{\text{ct}}$.*

**Proof:** We first show that $\mathcal{T}_G^{\text{ct}} \subseteq \mathcal{T}_G^{\text{rip}}$. Let $T_{\text{ct}} \in \mathcal{T}_G^{\text{ct}}$; choose $R \in \mathcal{K}_G$; and root $T_{\text{ct}}$ at $R$. Consider any reverse topological ordering $R = K_1, K_2, \ldots, K_m$ of the rooted tree $T_{\text{ct}}$. For any clique $K_j$, $2 \leq j \leq m$, let $K_p$ be its parent clique in the rooted tree (whence $1 \leq p \leq j-1$). Now, for $1 \leq i \leq j-1$, the clique $K_i$ cannot be a descendant of $K_j$, hence $K_p$ is on the path in $T_{\text{ct}}$ connecting $K_j$ and $K_i$. The clique-intersection property implies that $K_j \cap K_i \subseteq K_p$. This implies that $K_j \cap (K_1 \cup K_2 \cup \cdots \cup K_{j-1}) \subseteq K_p$; furthermore, $K_p$ cannot be a subset of $K_j$ by maximality, so the containment is proper. Thus, $T_{\text{ct}} \in \mathcal{T}_G^{\text{rip}}$, and we have $\mathcal{T}_G^{\text{ct}} \subseteq \mathcal{T}_G^{\text{rip}}$.

To see that $\mathcal{T}_G^{\text{rip}} \subseteq \mathcal{T}_G^{\text{ct}}$, consider a tree $T = (\mathcal{K}_G, \mathcal{E}) \notin \mathcal{T}_G^{\text{ct}}$. We will show that $T \notin \mathcal{T}_G^{\text{rip}}$. Since $T \notin \mathcal{T}_G^{\text{ct}}$, there exists then a pair of distinct cliques $K, K' \in \mathcal{K}_G$ such that the set $K \cap K'$ is not contained in at least one clique on the path connecting $K$ and $K'$ in the tree. Choose two such cliques $K, K' \in \mathcal{K}_G$ that minimize the length of the path from $K$ to $K'$ in $T$. The key observation on which our argument depends is that the set $K \cap K'$ belongs to *no* clique on the path connecting $K$ and $K'$ in the tree, except $K$ and $K'$. Let $K_1, K_2, \ldots, K_m$ be any reverse topological ordering of $T$ for arbitrary root $K_1 \in \mathcal{K}_G$. It suffices to show that (3.1) does not hold for some parent-child pair in $T$.

Consider the path $\mu = [K = K_{i_0}, K_{i_1}, \ldots, K_{i_s} = K']$ in $T$. Let $K_{i_t}$ be the clique with *lowest* index among the cliques in $\mu$, and without loss of generality assume that $i_0 > i_s$. Since under the given reverse topological ordering $K_{i_0}$ is a proper descendant of $K_{i_t} \in \mu$, the clique $K_{i_1}$ is necessarily the parent of $K_{i_0}$ in the rooted tree, and hence $i_0 > i_1$. Our choice of $K$ $(= K_{i_0})$ and $K'$ $(= K_{i_s})$ implies that (a) $s \geq 2$, and (b) $K_{i_0} \cap K_{i_s} \not\subseteq K_{i_r}$ for each $r$, $1 \leq r \leq s-1$. In consequence, we have $K_{i_0} \cap K_{i_s} \not\subseteq K_{i_1}$, whence (3.1) does not hold for the parent-child pair $K_{i_1}$ and $K_{i_0}$, which completes the proof. ∎

**Remark** In the preceding proof, the argument that $\mathcal{T}_G^{\text{ct}} \subseteq \mathcal{T}_G^{\text{rip}}$ verifies that any reverse topological ordering of a clique tree $T_{\text{ct}} \in \mathcal{T}_G^{\text{ct}}$ is an RIP ordering of the cliques.

### 3.4. The maximum-weight spanning tree property

Associated with each chordal graph $G$ is a *weighted clique intersection graph*, $W_G$, defined as follows. The vertex set of $W_G$ is the set of cliques $\mathcal{K}_G$. Two distinct cliques $K, K' \in \mathcal{K}_G$ are connected by an edge if and only if their intersection is nonempty; moreover, each such edge $(K, K')$ is assigned a positive weight given by $|K \cap K'|$. We let $\mathcal{T}_G^{\text{mst}}$ be the set containing every *maximum-weight spanning tree* (MST) of $W_G$.

Figure 3.4 shows $W_G$ for the chordal graph in Figure 3.1, and highlights the edges of the clique tree in Figure 3.2. Observe that the highlighted clique tree is a maximum-weight spanning tree of $W_G$, with edge weights that sum to five. Bernstein and Goodman [2] first showed that for any chordal graph $G$, we have $\mathcal{T}_G^{\text{mst}} = \mathcal{T}_G^{\text{ct}}$. Our proof of this result is similar to that given by Gavril [13].

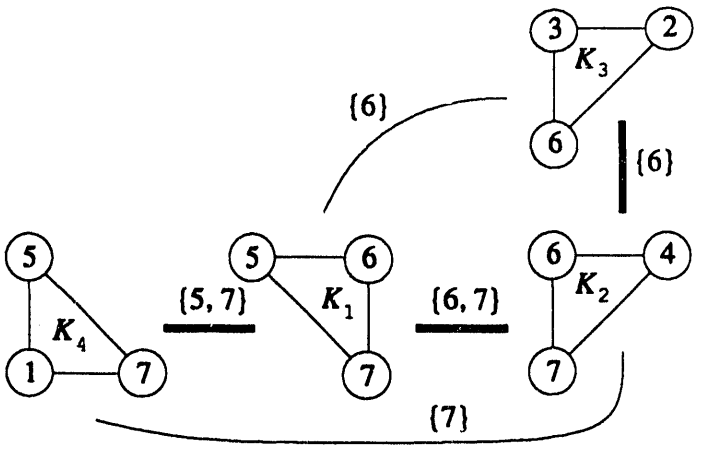


Figure 3.4: Weighted clique intersection graph for graph in Figure 3.1. Bold edges belong to the clique tree in Figure 3.2. Also shown are the intersection sets upon which the weights are based.

Our argument requires two ideas commonly used in the study of maximum-weight (minimum-weight) spanning tree algorithms. First, let $T = (\mathcal{K}_G, \mathcal{E}_T)$ be a spanning tree of $W_G$. It is well known that $T$ is a maximum-weight spanning tree if and only if for every pair of cliques $K, K' \in \mathcal{K}_G$ for which $(K, K') \notin \mathcal{E}_T$, the weight of every edge on the path joining $K$ and $K'$ in $T$ is no smaller than $|K \cap K'|$ (see, for example, Tarjan [44, pp. 71–72]). Second, given an edge $(K, K')$ in a tree, we define the *fundamental cut set* (see Gibbons [18, p. 58]) associated with the edge as follows. The removal of $(K, K')$ from the tree partitions the vertices of $T$ into precisely two sets, say $\mathcal{K}_1$ and $\mathcal{K}_2$. The fundamental cut set associated with $(K, K')$ consists of every edge with one vertex in $\mathcal{K}_1$ and the other in $\mathcal{K}_2$, including $(K, K')$ itself.

**Theorem 3.5 (Bernstein and Goodman [2]).** *For any connected chordal graph $G$, $\mathcal{T}_G^{\text{mst}} = \mathcal{T}_G^{\text{ct}}$.*

**Proof:** We first show that $\mathcal{T}_G^{\text{ct}} \subseteq \mathcal{T}_G^{\text{mst}}$. Let $T_{\text{ct}} \in \mathcal{T}_G^{\text{ct}}$ and choose two cliques $K$ and $K'$ that are not connected by an edge in $T_{\text{ct}}$. Consider the cycle formed by adding the edge $\{K, K'\}$ to $T_{\text{ct}}$. By Theorem 3.1 every edge along this cycle has weight no smaller than $|K \cap K'|$, whence $T_{\text{ct}}$ is a maximum-weight spanning tree of $W_G$.

To see that $\mathcal{T}_G^{\text{mst}} \subseteq \mathcal{T}_G^{\text{ct}}$, choose $T_{\text{mst}} \in \mathcal{T}_G^{\text{mst}}$. By Theorem 3.1, $\mathcal{T}_G^{\text{ct}} \neq \emptyset$. Choose $T_{\text{ct}} \in \mathcal{T}_G^{\text{ct}}$ that has a maximum number of edges in common with $T_{\text{mst}}$. Assume for the purpose of contradiction that there is an edge $(K_1, K_2)$ of $T_{\text{mst}}$ that is not an edge of $T_{\text{ct}}$. Consider the fundamental cut set (in $W_G$) associated with the edge $(K_1, K_2)$ of $T_{\text{mst}}$ and also the cycle (in $T_{\text{ct}}$) obtained by adding the edge $(K_1, K_2)$ to $T_{\text{ct}}$. Any cycle containing one edge from the cut set must contain another edge from the cut set as well. Select from the cycle in $T_{\text{ct}}$ one of the edges $(K_3, K_4) \neq (K_1, K_2)$ that belongs

to the cut set.

Note that the edge $(K_3, K_4)$ is an edge of $T_{ct}$, but it is not an edge of $T_{mst}$. Since $T_{ct}$ is a clique tree, it follows from Theorem 3.1 that $K_1 \cap K_2 \subseteq K_3 \cap K_4$. However, if $K_1 \cap K_2$ were a proper subset of $K_3 \cap K_4$, then replacing $(K_1, K_2)$ in $T_{mst}$ with $(K_3, K_4)$ would result in a spanning tree of greater weight, contrary to the maximality of $T_{mst}$'s weight. Hence, $K_1 \cap K_2 = K_3 \cap K_4$. Consider the tree obtained by replacing $(K_3, K_4)$ in $T_{ct}$ with the edge $(K_1, K_2)$. The reader can easily verify that the resulting tree is a clique tree. The new clique tree moreover has one more edge in common with $T_{mst}$ than originally possessed by $T_{ct}$, giving us the contradiction we seek. Consequently, $T_{mst} = T_{ct}$, and the result holds. ∎

### 3.5. Summary

The following corollary summarizes the results presented in this section.

**Corollary 1.** *For every connected graph $G$, we have*

$$\mathcal{T}_G^{ct} = \mathcal{T}_G^{ist} = \mathcal{T}_G^{rip}.$$

*Furthermore, $G$ is chordal if and only if this set is nonempty, in which case we have*

$$\mathcal{T}_G^{ct} = \mathcal{T}_G^{ist} = \mathcal{T}_G^{rip} = \mathcal{T}_G^{mst}.$$

Based on Corollary 1, we henceforth drop the superscripts from our notation and shall use $\mathcal{T}_G$ to denote the set of clique trees of $G$. Finally, Figure 3.5 illustrates Corollary 1 in negative form. We now verify that the tree displayed in this figure

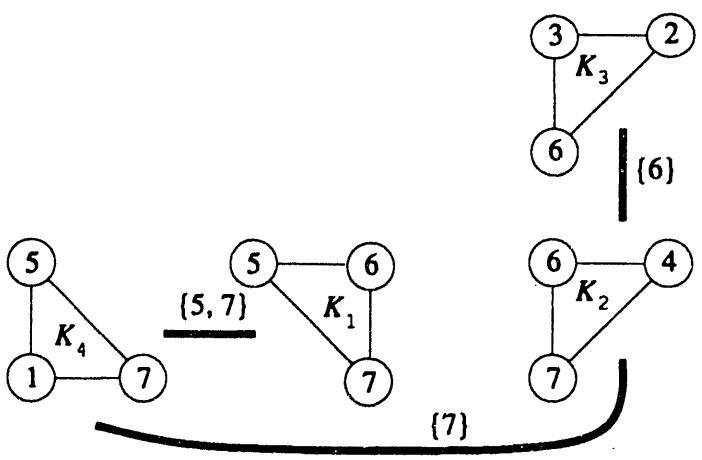


Figure 3.5: Not a clique tree of the graph in Figure 3.1.

indeed satisfies none of the characterizations of a clique tree:

[CT] The set $K_1 \cap K_2$ is not contained in $K_4$.

[IST] $\mathcal{K}_G(v_6)$ does not induce a subtree.

[RIP] The reverse topological ordering $K_3, K_2, K_4, K_1$ is not an RIP ordering: $K_1 \cap (K_4 \cup K_2 \cup K_3) = K_1$, which is, of course, contained in no other clique. It follows then from the remark after Theorem 5 that the tree is not an RIP tree.

[MST] The weight of the tree, which is four, is submaximal by one.

## 4. Clique trees, separators, and MCS revisited

This section ties together some of the results and concepts presented separately in Sections 2 and 3. Section 4.1 presents results that link the edges in a clique tree with the minimal vertex separators of the underlying chordal graph. Section 4.2 presents an efficient algorithm for computing a clique tree. This algorithm, which is a simple extension of the MCS algorithm, is shown to be an implementation of Prim's algorithm for finding a maximum-weight spanning tree of the weighted clique intersection graph $W_G$. New definitions and notation will be introduced as needed, and appropriate references to the literature will be given in each subsection. As in the previous section, we assume without loss of generality that $G$ is connected.

### 4.1. Clique tree edges and minimal vertex separators

Choose a clique tree $T \in \mathcal{T}_G$ and let $S = K_i \cap K_j$ for some edge $(K_i, K_j) \in \mathcal{E}_T$. Let $T_i = (\mathcal{K}_i, \mathcal{E}_i)$ and $T_j = (\mathcal{K}_j, \mathcal{E}_j)$ denote the two subtrees obtained by removing the edge $(K_i, K_j)$ from $T$, with $K_i \in \mathcal{K}_i$ and $K_j \in \mathcal{K}_j$. We also define vertex sets $V_i \subset V$ and $V_j \subset V$ by

$$V_i := \left( \bigcup_{K \in \mathcal{K}_i} K \right) - S$$

and

$$V_j := \left( \bigcup_{K \in \mathcal{K}_j} K \right) - S.$$

We first prove two technical lemmas, the second of which shows that the set $S = K_i \cap K_j$ separates $V_i$ from $V_j$ in $G$. These two results are then used in the proof of Theorem 4.1 to show that for any clique tree $T \in T_{ct}$ the set $S' \subset V$ is a minimal vertex separator if and only if $S' = K \cap K'$ for some edge $(K, K') \in \mathcal{E}_T$. The results in this section have appeared in both Ho and Lee [21] and Lundquist [33]. The proofs of Lemma 5 and Theorem 4.1 are similar to arguments given by Lundquist [33].

**Lemma 4.** *The sets $V_i$, $V_j$, and $S$ form a partition of $V$.*

**Proof:** Let $T$, $S$, $K_i$, $K_j$, $\mathcal{K}_i$, $\mathcal{K}_j$, $V_i$, and $V_j$ be as defined as in the first paragraph of the subsection. Clearly, $V = V_i \cup V_j \cup S$, and $S$ is disjoint from both $V_i$ and $V_j$. Hence it suffices to show that $V_i \cap V_j = \emptyset$. By way of contradiction assume the there exists a vertex $v \in V_i \cap V_j$. It follows that $v$ belongs to some clique $K \in \mathcal{K}_i$ and also belongs to some clique $K' \in \mathcal{K}_j$. Since $T \in \mathcal{T}_G$, the vertex $v$ belongs to every clique along the path joining $K$ and $K'$ in $T$, which necessarily includes both $K_i$ and $K_j$. In consequence, $v \in S = K_i \cap K_j$, which is impossible since both $V_i$ and $V_j$ are disjoint from $S$, whence the result follows. ∎

**Lemma 5.** *If $S = K_i \cap K_j$ and $(K_i, K_j) \in \mathcal{E}_T$ for some $T \in \mathcal{T}_G$, then $S$ is a $vw$-separator for every pair of vertices $v \in V_i$ and $w \in V_j$.*

**Proof:** Again let $T$, $S$, $K_i$, $K_j$, $\mathcal{K}_i$, $\mathcal{K}_j$, $V_i$, and $V_j$ be as defined in the first paragraph of the subsection. To prove the result it suffices to show that there exists no edge $(v, w) \in E_G$ with $v \in V_i$ and $w \in V_j$. Now, if $(v, w) \in E_G$, then there exists a clique $K \in \mathcal{K}_G$ for which $v, w \in K$. If $K \in \mathcal{K}_i$ then clearly $v, w \in S \cup V_i$. Moreover since by Lemma 4, $V_i$, $V_j$, and $S$ form a partition of $V$, it follows that neither $v$ nor $w$ belongs to $V_j$. Likewise, if $K \in \mathcal{K}_j$ then $v, w \in S \cup V_j$, and neither $v$ nor $w$ belongs to $V_i$. In consequence, no edge in $E_G$ joins two vertices $v \in V_i$ and $w \in V_j$, which concludes the proof. ∎

**Theorem 4.1.** *Let $T \in \mathcal{T}_G$. The set $S \subset V$ is a minimal vertex separator of $G$ if and only if $S = K \cap K'$ for some edge $(K, K') \in \mathcal{E}_T$.*

**Proof:** For the "if" part let $T \in \mathcal{T}_G$, and let $S = K \cap K'$, for some edge $(K, K') \in \mathcal{E}_T$. Consider two vertices $v \in K - S$ and $w \in K' - S$. By Lemma 5, $S$ is a $vw$-separator. Moreover, since both $v$ and $w$ are adjacent to every vertex in $S$, it follows that $S$ is a minimal $vw$-separator, as desired.

To prove the "only if" part, choose $T \in \mathcal{T}_G$ and let $S$ be a minimal $vw$-separator of $G$. Since $(v, w) \notin E$, the sets $\mathcal{K}_G(v)$ and $\mathcal{K}_G(w)$ induce disjoint subtrees of $T$. Choose $K \in \mathcal{K}_G(v)$ and $K' \in \mathcal{K}_G(w)$ to minimize the distance in $T$ between $K$ and $K'$. Consider the path $\mu = [K = K_0, K_1, \ldots, K_{r-1}, K_r = K']$ in $T$, where $r \geq 1$. Define $S_i := K_i \cap K_{i+1}$ for $0 \leq i \leq r - 1$, and let $\mathcal{S} := \{S_0, S_1, \ldots, S_{r-1}\}$. We will show that $S \in \mathcal{S}$, which suffices to prove the result.

First, to see that $S_i \subseteq S$ for at least one set $S_i \in \mathcal{S}$, suppose (for the purpose of contradiction) that $S_i \not\subseteq S$ for every $S_i \in \mathcal{S}$, and choose $x_i \in S_i - S$ for each member of $\mathcal{S}$. Since $x_i \in K_i \cap K_{i+1}$ $(0 \leq i \leq r - 1)$, we have a path $[v, x_0, x_1, \ldots, x_{r-1}, w]$ joining $v$ and $w$ in $G \setminus S$, contrary to our assumption that $S$ is a $vw$-separator. It follows that $S_i \subseteq S$ for at least one set $S_i \in \mathcal{S}$.

Now select $S_i \in \mathcal{S}$ for which $S_i \subseteq S$, and consider the two subtrees obtained by removing the edge $(K_i, K_{i+1})$ from $T$. Let $T_v$ be the subtree containing $K_0 \ni v$, and let $T_w$ be the subtree containing $K_r \ni w$. Since $S_i$ is contained in the $vw$-separator $S$, we clearly have $v, w \notin S_i$. Hence, by Lemma 5, $S_i$ is a $vw$-separator. Since $S$ is

moreover a minimal $vw$-separator, we have $S = S_i = K_i \cap K_{i+1}$ where $(K_i, K_{i+1}) \in \mathcal{E}_T$, as required. ∎

For a clique tree $T = (\mathcal{K}_G, \mathcal{E}_T) \in \mathcal{T}_G$, consider the set containing every *distinct* set $K \cap K'$ where $(K, K') \in \mathcal{E}_T$. It follows immediately from Theorem 4.1 that this set is the same for every clique tree $T \in \mathcal{T}_G$. In light of Theorem 4.1, we shall refer to the members of this invariant set as *separators*. For any clique tree $T = (\mathcal{K}_G, \mathcal{E}_T) \in \mathcal{T}_G$ consider the **multiset** of separators defined by

$$\mathcal{M}_T := \{K \cap K' \mid (K, K') \in \mathcal{E}_T\}.$$

That this multiset is the same for all clique trees $T \in \mathcal{T}_G$ is an immediate consequence of a result by Ho and Lee [21]; the result was also proven by Lundquist [33]. The proof is taken directly from Blair and Peyton [4].

**Theorem 4.2 (Ho and Lee [21], Lundquist [33]).** *The multiset of separators is the same for every clique tree $T \in \mathcal{T}_G$.*

**Proof:** For the purpose of contradiction, suppose there exist two distinct clique trees $T, T' \in \mathcal{T}_G$ for which $\mathcal{M}_T \neq \mathcal{M}_{T'}$. From among the clique trees $T' \in \mathcal{T}_G$ for which $\mathcal{M}_{T'} \neq \mathcal{M}_T$, choose $T'$ so that it shares as many edges as possible with $T$. (Note that $T$ and $T'$ cannot share the same edge set, for then they also would share the same multiset of separators.)

Let $(K_1, K_2)$ be an edge of $T$ that does not belong to $T'$. As in the proof of Theorem 3.5, consider the fundamental cut set (in $W_G$) associated with the edge $(K_1, K_2)$ of $T$ and also the cycle (in $T'$) obtained by adding the edge $(K_1, K_2)$ to $T'$. Recall that any cycle containing one edge from the cut set must contain another edge from the cut set as well. Select from the cycle in $T'$ one of the edges $(K_3, K_4) \neq (K_1, K_2)$ that belongs to the cut set. Note that the edge $(K_3, K_4)$ is an edge of $T'$ but not an edge of $T$.

Since $T \in T_{\text{ct}}$, it follows by Theorem 3.1 that $K_3 \cap K_4 \subseteq K_1 \cap K_2$; similarly, since $T' \in T_{\text{ct}}$, it follows by Theorem 3.1 that $K_1 \cap K_2 \subseteq K_3 \cap K_4$; hence $K_3 \cap K_4 = K_1 \cap K_2$. By Theorem 3.5, the replacement of $(K_3, K_4)$ in $T'$ with $(K_1, K_2)$ results in a clique tree, which, moreover, clearly has the same multiset of separators that $T'$ has. Contrary to our assumption about $T'$, the modified tree shares one more edge with $T$, and thus result follows. ∎

## 4.2. MCS and Prim's algorithm

Prim's algorithm [38] is an efficient method for computing a maximum-weight (minimum-weight) spanning tree of a weighted graph. Thus, by Theorem 3.5, Prim's algorithm applied to the weighted clique intersection graph $W_G$ computes a clique tree $T \in \mathcal{T}_G$. At any point the algorithm has constructed a subtree of the eventual maximum-weight

spanning tree $T$, and at each step it adds one more clique and edge to this subtree. Let $\hat{\mathcal{K}} \subset \mathcal{K}_G$ be the cliques in the subtree constructed thus far. As the next edge to be added, the algorithm chooses the heaviest edge that joins $\hat{\mathcal{K}}$ to $\mathcal{K}_G - \hat{\mathcal{K}}$. For a proof that Prim's algorithm correctly computes a maximum-weight spanning tree, we refer the reader to Tarjan [44, pp. 73–75] or Gibbons [18, pp. 40–42]. A version of Prim's algorithm formulated specifically for our problem is given in Figure 4.1.

---

$\mathcal{E}_T \leftarrow \emptyset$;
Choose $K \in \mathcal{K}_G$;
$\hat{\mathcal{K}} \leftarrow \{K\}$;
**for** $r \leftarrow 2$ **to** $m$ **do**
    Choose cliques $K \in \hat{\mathcal{K}}$ and $K' \in \mathcal{K}_G - \hat{\mathcal{K}}$
        for which $|K \cap K'|$ is maximum;
    $\mathcal{E}_T \leftarrow \mathcal{E}_T \cup \{(K, K')\}$;
    $\hat{\mathcal{K}} \leftarrow \hat{\mathcal{K}} \cup \{K'\}$;
**end for**

Figure 4.1: Prim's algorithm for finding a maximum-weight spanning tree of the weighted clique intersection graph $W_G$.

---

In this section we will show that the MCS algorithm applied to a chordal graph $G$ can be viewed as an implementation of Prim's algorithm applied to $W_G$. In Section 4.2.1 we show that since the MCS algorithm generates a PEO, it can easily detect the cliques in $\mathcal{K}_G$ during the course of the computation. Section 4.2.2 shows that 1) the MCS algorithm can be viewed as a block algorithm that "searches" the cliques in $\mathcal{K}_G$ one after the other, and 2) the order in which the cliques are searched is precisely the order in which the cliques are searched by Prim's algorithm in Figure 4.1. Using the results in Sections 4.2.1 and 4.2.2, we also show how to supplement the MCS algorithm with a few additional statements so that it detects the cliques and a set of clique tree edges as it generates a PEO. A detailed statement of this algorithm appears at the end of Section 4.2.2.

The close connection between the MCS algorithm and Prim's algorithm was, to our knowledge, first presented by Blair, England, and Thomason [3]. Several of the proofs in this section are similar to arguments given by Lewis *et al.* [24]. Though the techniques discussed in this section can be implemented to run quite efficiently, there are more efficient ways to compute a clique tree when certain data structures that arise in sparse matrix computations are available. The reader should consult Lewis et al. [24] for details on how to compute a clique tree in the course of solving a sparse positive definite linear system.

### 4.2.1. Detecting the cliques

In this subsection we show that the MCS algorithm can easily and efficiently detect the cliques in $\mathcal{K}_G$. To do so we exploit the fact that MCS computes a PEO. We shall use the following result from Fulkerson and Gross [10].

**Lemma 6 (Fulkerson and Gross [10]).** *Let $v_1, v_2, \ldots, v_n$ be a perfect elimination ordering of $G$. The set of maximal cliques $\mathcal{K}_G$ contains precisely the sets $\{v_i\} \cup madj(v_i)$ for which there exists no vertex $v_j$, $j < i$, such that*

$$\{v_i\} \cup madj(v_i) \subset \{v_j\} \cup madj(v_j). \tag{4.1}$$

**Proof:** Choose $K \in \mathcal{K}_G$ and let $v_i \in K$ be the vertex whose label $i$ assigned by the PEO is lowest among the labels assigned to a vertex of $K$. Consider the vertex set $\{v_i\} \cup madj(v_i)$. Since $K$ consists of $v_i$ and neighbors of $v_i$ with labels larger than $i$, clearly $K \subseteq \{v_i\} \cup madj(v_i)$. Because the ordering is a PEO, the set $\{v_i\} \cup madj(v_i)$ must be complete in $G$. Thus by maximality of the clique $K$ we have $K = \{v_i\} \cup madj(v_i)$, and moreover it follows that (4.1) holds for no vertex $v_j$, $j < i$.

Now, let $K = \{v_i\} \cup madj(v_i)$ and suppose that (4.1) holds for no vertex $v_j$, $j < i$. Since the ordering is a PEO, clearly $K$ is complete in $G$. If $K$ is submaximal, then there exists a vertex $v_j \in V - K$ that is adjacent to every vertex of $K$. But the existence of such a vertex $v_j$ is impossible: if $j > i$ then $v_j \in madj(v_i)$, contrary to $v_j \in V - K$; if $j < i$ then (4.1) holds for $v_j$, contrary to our assumption. In consequence, no such vertex $v_j$ exists, and the result follows. ∎

Throughout the remainder of the paper we let $v_1, v_2, \ldots, v_n$ be a PEO obtained by applying the MCS algorithm to a connected chordal graph $G$. We shall call $v_{i_r}$ the *representative vertex* of $K_r$ whenever $K_r = \{v_{i_r}\} \cup madj(v_{i_r})$; that is, we let $v_{i_1}, v_{i_2}, \ldots, v_{i_m}$ be the representative vertices of the cliques $K_1, K_2, \ldots, K_m$, respectively, where $i_1 > i_2 > \cdots > i_m$. Thus the ordering $K_1, K_2, \ldots, K_m$ specifies the order in which the cliques are searched by the MCS algorithm.

As the MCS algorithm generates a PEO it can easily detect the representative vertices and hence can easily collect the cliques in $\mathcal{K}_G$. Condition 2 in the next lemma provides a test for determining when a vertex in an MCS ordering is *not* a representative vertex. Lemma 8 then provides a simple test for detecting the representative vertices.

**Lemma 7.** *Let $v_1, v_2, \ldots, v_n$ be a perfect elimination ordering obtained by applying the maximum cardinality search algorithm to a connected chordal graph $G$. For each vertex label $i$, $1 \leq i \leq n-1$, the following are equivalent:*

1. $\{v_{i+1}\} \cup madj(v_{i+1}) \notin \mathcal{K}_G$.
2. $|adj(v_i) \cap \mathcal{L}_{i+1}| = |adj(v_{i+1}) \cap \mathcal{L}_{i+2}| + 1$.
3. $\{v_i\} \cup madj(v_i) = \{v_i, v_{i+1}\} \cup madj(v_{i+1})$.

**Proof:** First we state two inequalities that prove useful here and in later proofs. Note that the maximum cardinality selection criterion ensures that the following inequality holds true when $v_{i+1}$ $(1 \leq i \leq n-1)$ is selected to be labeled:

$$|adj(v_i) \cap \mathcal{L}_{i+2}| \leq |adj(v_{i+1}) \cap \mathcal{L}_{i+2}|. \tag{4.2}$$

Equation (4.2) along with the fact that $\mathcal{L}_{i+1} = \mathcal{L}_{i+2} \cup \{v_{i+1}\}$, gives us

$$|adj(v_i) \cap \mathcal{L}_{i+1}| \leq |adj(v_{i+1}) \cap \mathcal{L}_{i+2}| + 1. \tag{4.3}$$

Assume that the first condition in the statement of the lemma holds for $v_{i+1}$, and consider the vertex $v_i$ selected by the MCS algorithm at the next step. When the algorithm selects $v_i$ there exists (by Lemma 6) a vertex $u \in V - \mathcal{L}_{i+1}$ that is adjacent to every vertex in $\{v_{i+1}\} \cup madj(v_{i+1})$. In light of (4.3), the existence of such a vertex $u$ ensures that the vertex $v_i$ chosen by the MCS algorithm (perhaps $v_i = u$) satisfies the second condition.

Assume now that the second condition in the statement of the lemma holds for the two vertices $v_i$ and $v_{i+1}$. It immediately follows that

$$|madj(v_i)| = |\{v_{i+1}\} \cup madj(v_{i+1})|.$$

Consequently, to prove that the third condition holds true it suffices to show that $madj(v_i) \subseteq \{v_{i+1}\} \cup madj(v_{i+1})$. Now if it were the case that $v_{i+1} \notin adj(v_i)$, then from (4.2) and the fact that $\mathcal{L}_{i+1} = \mathcal{L}_{i+2} \cup \{v_{i+1}\}$ we would have

$$|adj(v_i) \cap \mathcal{L}_{i+1}| \leq |adj(v_{i+1}) \cap \mathcal{L}_{i+2}|,$$

contrary to our assumption that condition 2 holds true. It follows then that $v_{i+1}$ is adjacent to $v_i$ in $G$. Now choose $v_k \in madj(v_i) - \{v_{i+1}\}$. Clearly $k \geq i+2$; moreover, since $\{v_i\} \cup madj(v_i)$ is complete in $G$, $v_k$ is necessarily adjacent to $v_{i+1} \in madj(v_i)$; whence $v_k \in madj(v_{i+1})$, giving us condition 3.

Finally, by Lemma 6 the first condition follows immediately from the third, which completes the proof. ∎

Further extending the result in Lemma 7, we obtain the following technique for detecting the representative vertices of $\mathcal{K}_G$ while generating the MCS ordering.

**Lemma 8.** *Let $v_1, v_2, \ldots, v_n$ be a perfect elimination ordering obtained by applying the maximum cardinality search algorithm to a connected chordal graph $G$. Then $\mathcal{K}_G$ contains precisely the following sets: $\{v_1\} \cup madj(v_1)$ and $\{v_{i+1}\} \cup madj(v_{i+1})$, $1 \leq i \leq n-1$, for which*

$$|adj(v_i) \cap \mathcal{L}_{i+1}| \leq |adj(v_{i+1}) \cap \mathcal{L}_{i+2}|. \tag{4.4}$$

**Proof:** From Lemma 6 it follows that $\{v_1\} \cup madj(v_1) \in \mathcal{K}_G$. Consider the set $\{v_{i+1}\} \cup madj(v_{i+1})$ where $1 \leq i \leq n-1$. It follows from (4.3) and the equivalence of conditions 1 and 2 in Lemma 7 that $\{v_{i+1}\} \cup madj(v_{i+1})$ is a member of $\mathcal{K}_G$ if and only if (4.4) holds. This concludes the proof. ∎

### 4.2.2. MCS as a block algorithm

Clearly, the MCS algorithm can detect the cliques in $\mathcal{K}_G$ by determining at each step whether or not (4.4) holds. With the next lemma we show that the MCS algorithm can be viewed as a block algorithm that searches the cliques of $\mathcal{K}_G$ one after the other.

**Lemma 9.** *Let $v_1, v_2, \ldots, v_n$ be a perfect elimination ordering obtained by applying the maximum cardinality search algorithm to a connected chordal graph $G$, and let $v_{i_1}, v_{i_2}, \ldots, v_{i_m}$ be the representative vertices of the cliques $K_1, K_2, \ldots, K_m$, respectively, where $i_1 > i_2 > \cdots > i_m$. Then*

$$\mathcal{L}_{i_r} = \bigcup_{s=1}^{r} K_s \tag{4.5}$$

*for each $r$, $1 \leq r \leq m$.*

**Proof:** Choose $r$, $1 \leq r \leq m$, and assume $v_j \notin \mathcal{L}_{i_r}$, i.e., $j < i_r$. Since clearly $v_j \notin \{v_{i_s}\} \cup madj(v_{i_s})$ for each $s$, $1 \leq s \leq r$, it follows by Lemma 6 that $v_j \notin \cup_{s=1}^{r} K_s$. Now assume $v_j \in \mathcal{L}_{i_r}$ and for convenience of notation define $i_0 := n+1$. Choose $s$, $1 \leq s \leq r$, for which $i_s \leq j < i_{s-1}$. If $j = i_s$, then clearly $v_j \in K_s = \{v_j\} \cup madj(v_j)$. If $i_s < j$, then by repeated application of condition 3 of Lemma 7, we have

$$\begin{aligned} K_s &= \{v_{i_s}\} \cup madj(v_{i_s}), \\ &= \{v_{i_s}, v_{i_s+1}\} \cup madj(v_{i_s+1}), \\ &\vdots \\ &= \{v_{i_s}, v_{i_s+1}, \ldots, v_j\} \cup madj(v_j). \end{aligned}$$

Consequently, $v_j \in K_s$, and the result follows. ∎

It follows from Lemma 9 that the MCS algorithm labels the vertices contiguously in blocks as follows:

$$\begin{aligned} \{v_{i_1}, v_{i_1+1}, \ldots, v_n = v_{i_0-1}\} &= K_1 \\ \{v_{i_2}, v_{i_2+1}, \ldots, v_{i_1-1}\} &= K_2 - K_1 \\ \{v_{i_3}, v_{i_3+1}, \ldots, v_{i_2-1}\} &= K_3 - \bigcup_{s=1}^{2} K_s \\ &\vdots \end{aligned}$$

$$\{v_1 = v_{i_m}, v_{i_m+1}, \ldots, v_{i_{m-1}-1}\} \;=\; K_m - \bigcup_{s=1}^{m-1} K_s.$$

For convenience we define the function *clique* : $V \rightarrow \{1, \ldots, m\}$ by $clique(v_j) := r$ where $i_0 := n + 1$ and $v_j \in \{v_{i_r}, v_{i_r+1}, \ldots, v_{i_{r-1}-1}\}$ (i.e., $i_r \leq j < i_{r-1}$). Clearly $clique(v)$ is the lowest index of a clique that contains $v$; that is,

$$clique(v) = \min\{r \mid v \in K_r\}.$$

The following lemma is needed to provide a means of detecting the edges of a clique tree, and it is also critical in the proof of the main result in this subsection.

**Lemma 10.** *Let $v_1, v_2, \ldots, v_n$ be a perfect elimination ordering obtained by applying the maximum cardinality search algorithm to a connected chordal graph $G$, and let $v_{i_1}, v_{i_2}, \ldots, v_{i_m}$ be the representative vertices of the cliques $K_1, K_2, \ldots, K_m$, respectively, where $i_1 > i_2 > \cdots > i_m$. For any integer $r$, $1 \leq r \leq m - 1$, there exists an integer $p$, $1 \leq p \leq r$, such that*

$$K_{r+1} \cap \mathcal{L}_{i_r} = K_{r+1} \cap K_p. \tag{4.6}$$

*Moreover, Equation (4.6) is satisfied when $p = clique(v_j)$, where $v_j$ is the vertex in $K_{r+1} \cap \mathcal{L}_{i_r}$ with smallest label $j$.*

**Proof:** Let $1 \leq r \leq m - 1$. From Lemma 9 it follows that for $1 \leq p \leq r$ we have

$$K_{r+1} \cap K_p \subseteq K_{r+1} \cap \mathcal{L}_{i_r}.$$

To prove the result it suffices to show that $K_{r+1} \cap \mathcal{L}_{i_r} \subseteq K_p$. Now consider the set $K_{r+1} \cap \mathcal{L}_{i_r}$, and choose $v_j \in K_{r+1} \cap \mathcal{L}_{i_r}$ with smallest label $j$. Clearly $K_{r+1} \cap \mathcal{L}_{i_r}$ is complete in $G$ and moreover

$$K_{r+1} \cap \mathcal{L}_{i_r} \subseteq \{v_j\} \cup madj(v_j). \tag{4.7}$$

Choose $p$, $1 \leq p \leq r$, for which $i_p \leq j < i_{p-1}$. (Note that $p = clique(v_j)$.) By the same argument used in the proof of Lemma 9, we have

$$\{v_j\} \cup madj(v_j) \subseteq K_p. \tag{4.8}$$

Combining (4.7) and (4.8), we obtain the result. ∎

From Lemmas 9 and 10 it follows that any MCS clique ordering is also an RIP ordering. Furthermore, Lemma 10 shows specifically how to use the *clique* function to obtain the edges of a clique tree in an efficient manner. (This technique for determining a clique tree parent function was introduced by Tarjan and Yannakakis [45] and also

appears in Lewis et al. [24].) It follows that the MCS algorithm can generate a clique tree by 1) detecting the cliques via representative vertices (Lemma 8) and 2) choosing as the parent of $K_{r+1}$ the clique $K_p$ for which $p = clique(v_j)$ where $j$ is the smallest label in $K_{r+1} \cap \mathcal{L}_{i_r}$. The following result shows that any clique tree generated in this fashion could also be generated by Prim's algorithm applied to $W_G$.

**Theorem 4.3.** *Any order in which the cliques are searched by the maximum cardinality search algorithm is also an order in which the cliques are searched by Prim's algorithm applied to* $W_G$.

**Proof:** Let $K_1, K_2, \ldots, K_m$ be an ordering of $\mathcal{K}_G$ generated by the MCS algorithm. Choose $r$, $1 \leq r \leq m-1$. To show that this clique ordering is also a search order for Prim's algorithm applied to $W_G$ (see Figure 4.1), it suffices to show that there exists $p$ $(1 \leq p \leq r)$ for which

$$|K_{r+1} \cap K_p| = \max_{\substack{1 \leq s \leq r \\ r+1 \leq t \leq m}} |K_t \cap K_s|. \tag{4.9}$$

To prove that (4.9) holds, choose any $s$ and $t$ for which $1 \leq s \leq r < t \leq m$. Consider the vertex $v_j \in K_{r+1} \cap \mathcal{L}_{i_r}$ for which $j$ is minimum, and let $p = clique(v_j)$. By Lemma 10, we can write

$$K_{r+1} \cap K_p = K_{r+1} \cap \mathcal{L}_{i_r}. \tag{4.10}$$

Lemma 9 and the discussion following that result imply that $v_{i_r-1}$ is the vertex from $K_{r+1} - \mathcal{L}_{i_r}$ whose label is maximum. By repeated application of condition 3 of Lemma 7 (as needed) we obtain the following:

$$\begin{aligned} K_{r+1} &= \{v_{i_{r+1}}\} \cup madj(v_{i_{r+1}}), \\ &= \{v_{i_{r+1}}, v_{i_{r+1}+1}\} \cup madj(v_{i_{r+1}+1}), \\ &\vdots \\ &= \{v_{i_{r+1}}, \ldots, v_{i_r-2}, v_{i_r-1}\} \cup madj(v_{i_r-1}). \end{aligned}$$

In consequence we have

$$K_{r+1} \cap \mathcal{L}_{i_r} = adj(v_{i_r-1}) \cap \mathcal{L}_{i_r}. \tag{4.11}$$

Now, if

$$|K_t \cap \mathcal{L}_{i_r}| > |adj(v_{i_r-1}) \cap \mathcal{L}_{i_r}|,$$

then for $u \in K_t - \mathcal{L}_{i_r} \neq \emptyset$, we have

$$|adj(v_{i_r-1}) \cap \mathcal{L}_{i_r}| < |adj(u) \cap \mathcal{L}_{i_r}|,$$

contrary to the maximum cardinality search criterion by which the vertices were labeled. It follows then that

$$|adj(v_{i_r-1}) \cap \mathcal{L}_{i_r}| \geq |K_t \cap \mathcal{L}_{i_r}|. \tag{4.12}$$

Finally, Lemma 9 implies that

$$|K_t \cap \mathcal{L}_{i_r}| \geq |K_t \cap K_s|. \tag{4.13}$$

Combining (4.10), (4.11), (4.12), and (4.13) shows that (4.9) holds, giving us the result. ∎

From the results in this subsection, we obtain an expanded version of the MCS algorithm, which computes a clique tree in addition to a PEO. The MCS algorithm is shown in Figure 2.3, and the expanded algorithm is shown in Figure 4.2. We emphasize

---

```
prev_card ← 0;
L_{n+1} ← ∅;
s ← 0;
E_T ← ∅;
for i ← n to 1 step −1 do
    Choose a vertex v ∈ V − L_{i+1} for which
        |adj(v) ∩ L_{i+1}| is maximum;
    α(v) ← i;  [v becomes v_i]
    new_card ← |adj(v_i) ∩ L_{i+1}|;
    if new_card ≤ prev_card then    [begin new clique]
        s ← s + 1;
        K_s ← adj(v_i) ∩ L_{i+1};  [= madj(v)]
        if new_card ≠ 0 then    [get edge to parent]
            k ← min{j | v_j ∈ K_s};
            p ← clique(v_k);
            E_T ← E_T ∪ {K_s, K_p};
        end if
    end if
    clique(v_i) ← s;
    K_s ← K_s ∪ {v_i};
    L_i ← L_{i+1} ∪ {v_i};
    prev_card ← new_card;
end for
```

Figure 4.2: An expanded version of MCS, which implements Prim's algorithm in Figure 4.1.

---

that the primary purpose of this section is to establish the connection between the MCS

algorithm and Prim's algorithm (applied to $W_G$), and Theorem 4.3 demonstrates that the detailed algorithm in Figure 4.2 can be viewed as a special implementation of Prim's algorithm shown in Figure 4.1. Some of the details necessary to represent a chordal graph as a clique tree have been discussed here; for a complete discussion of this topic the reader should consult the papers [24,45]. It is worth noting that a clique tree is often a much more compact and more computationally efficient data structure than the adjacency lists usually used to represent $G$.

## 5. Applications

In this section we briefly review a few recent applications of chordal graphs and clique trees in sparse matrix computations.

### 5.1. Terminology

Let $Ax = b$ be a sparse symmetric positive definite system of linear equations, whose Cholesky factorization is denoted by $A = LL^T$. Direct methods for solving such linear systems store and compute only the nonzero entries of the Cholesky factor $L$. This factorization generally introduces *fill* (or *fill-in*) into the matrix; that is, some of the zero entries in $A$ become nonzero entries in $L$.

Assume the coefficient matrix $A$ is $n \times n$. We associate a graph $G_A = (V, E_A)$ with the matrix $A$ in the usual way: the vertex set is given by $V = \{v_1, v_2, \ldots, v_n\}$, with two vertices $v_i$ and $v_j$ joined by an edge in $E_A$ if and only if $a_{ij} \neq 0$. We define the *filled graph* $G_F = (V, E_F)$ in precisely the same way, where $F := L + L^T$. Note that $G_F$ is a *chordal supergraph* of $G_A$ ($E_A \subseteq E_F$) [39], and the order in which the unknowns are eliminated is a PEO for the corresponding filled graph $G_F$.

### 5.2. Elimination trees

More commonly used than the clique tree, the *elimination tree* associated with the ordered graph $G_A$ has proven very useful in sparse matrix computations. The elimination tree $T_A = (V, E_T)$ for an irreducible graph $G_A$ is a rooted tree defined by a parent function as follows: for each vertex $v_j$, $1 \leq j \leq n-1$, the parent of $v_j$ is $v_i$, where the first off-diagonal nonzero entry in column $j$ of $L$ occurs in row $i > j$. If $G_A$ is reducible, one obtains a forest rather than a tree. A *topological ordering* of $T_A$ is any ordering of the vertices that numbers each parent with a label larger than that of any of its children. The order in which the unknowns are eliminated, for example, is a topological ordering of the tree $T_A$, and, in fact, any topological ordering of the tree is a PEO of $G_F$. Elimination trees evidently were introduced by Schreiber [41], though they had earlier been used implicitly in a number of algorithms and applications. Liu [31] has provided a survey of the many uses of elimination trees in sparse matrix computations.

Liu has also discovered an interesting connection between clique trees and elimination trees. To facilitate our discussion of this connection we need to introduce the following concepts and results. If $\mathcal{F}$ is a finite family of nonempty sets, then the *intersection graph* of $\mathcal{F}$ is obtained by representing each set in $\mathcal{F}$ by a vertex and connecting two vertices by an edge if and only if the intersection of the corresponding sets is nonempty. A *subtree graph* is an intersection graph where $\mathcal{F}$ is a family of subtrees of a specific tree. Buneman [5], Gavril [12], and Walter [46] independently discovered that the set of chordal graphs coincides with the set of subtree graphs in a result that further extends Theorem 3.3.

Theorem 3.2 provides an obvious way to represent a chordal graph $G := G_F$ as a subtree graph. Choose any clique tree $T_{\text{ct}} \in \mathcal{T}_G$, and consider the family of subtrees of $T_{\text{ct}}$ given by

$$\mathcal{F} = \{\mathcal{K}_G(v) \mid v \in V\}.$$

Since two vertices are adjacent to one another in $G$ if and only if there exists a clique $K \in \mathcal{K}_G$ to which both vertices belong, it follows that for each pair of vertices $u, v \in V$, we have $(u, v) \in E$ if and only if the subtree induced by $\mathcal{K}_G(u)$ intersects the subtree induced by $\mathcal{K}_G(v)$. In consequence, $G$ is a subtree graph for the family of subtrees $\mathcal{F}$ in any clique tree $T_{\text{ct}} \in \mathcal{T}_G$.

Liu has shown how elimination trees provide another way to view chordal graphs as subtree graphs. Let the *row vertex set*, denoted $Struct(L_{i,*})$, be defined by

$$Struct(L_{i,*}) := \{v_j \mid \ell_{ij} \neq 0\}.$$

Liu [27] has shown that each row vertex set $Struct(L_{i,*})$ induces a subtree of $T_A$ rooted at $v_i$. In consequence, $G_F$ is a subtree graph for the family of subtrees induced by the row vertex sets of $L$. For a full discussion of this result, consult Liu [31].

### 5.3. Equivalent orderings

The fill added to $G_A$ contains precisely the edges needed to make the order in which the unknowns of the linear system are eliminated a PEO of the filled graph $G_F$ [39]. Usually, the primary objective in reordering the linear system is to reduce the storage (i.e., fill) and work required by the factorization. Every PEO of $G_F$ results in precisely the same factorization storage and work requirement [29]. It is common practice in this setting to define all perfect elimination orderings of $G_F$ as *equivalent orderings.*

Before advanced machine architectures entered the marketplace, there was little reason to consider choosing one PEO of $G_F$ over another. Generally, whatever ordering was produced by the fill-reducing ordering algorithm (e.g., nested dissection [14,15] or minimum degree [17,26]) was accepted without modification. But this situation has changed to some extent with the advent of vector supercomputers, powerful RISC-based workstations, and a wide variety of parallel architectures. Algorithms designed for such

machines may benefit by choosing one PEO of $G_F$ over the others in order to optimize some secondary objective function. (There is still the underlying assumption that a good fill-reducing ordering is desired, though this assumption is subject to question more than it once was and deserves further study.) The following summarizes a few algorithms designed to produce an equivalent ordering that optimizes some secondary objective function.

**Reordering for stack storage reduction** One of the first algorithms of this type was a simple algorithm due to Liu [28] for finding, among all topological orderings of the elimination tree, an ordering that minimizes the auxiliary storage required by the multifrontal factorization algorithm. In addition, Liu [29] gives a heuristic for finding an equivalent ordering that further reduces auxiliary storage for multifrontal factorization. Finding an optimal equivalent ordering for this problem is still an open question.

**Jess and Kees reordering** Short elimination trees can be useful when the factorization is to be performed in parallel. Jess and Kees [22] introduced a simple greedy heuristic for finding an equivalent ordering that reduces elimination tree height. Liu [30] has shown that the Jess and Kees ordering scheme minimizes elimination tree height among all equivalent orderings. Liu and Mirzaian [25] introduced an $O(n+|E_F|)$ implementation of the Jess and Kees scheme. Lewis, Peyton, and Pothen [24] used a clique tree of $G_F$ to obtain an $O(n+q)$-time implementation of the Jess and Kees algorithm where $q = \sum_{i=1}^{m} |K_i|$, which in practice is substantially smaller than $|E_F|$. Because a PEO of $G_F$ is known *a priori*, a clique tree of $G_F$ can be obtained in $O(n)$ time using output from the symbolic factorization step of the solution process [24].

**A block Jess and Kees reordering** Blair and Peyton [4] have studied a block form of the Jess and Kees algorithm that generates a clique tree $T \in \mathcal{T}_G$ of minimum diameter. The primary motivation for this algorithm is to minimize the number of expensive communication calls to the general router on a fine-grained parallel machine [19]. The time complexity of their algorithm is also $O(n+q)$ in the sparse matrix setting, where a PEO is known *a priori*. A similar algorithm motivated by the same application was given by Gilbert and Schreiber [19].

**Partitioning (and reordering) for parallel triangular solution** A related problem is the following: Find a partition of the columns in the factor $L$ with as few members as possible, such that for each partition member, the elementary elimination matrices associated with that member can be multiplied together without increasing the storage requirement for the factor. Such a partition and its associated PEO is desirable for implementing sparse triangular solution on a fine-grained massively parallel machine. Pothen and Alvarado [37] have solved this problem when the ordering is restricted to topological orderings of the elimination tree. Peyton, Pothen, and Yuan [36] have

developed an $O(n + |E_F|)$ algorithm that solves the problem for the larger set of all equivalent orderings; they are also working on an $O(n + q)$ clique-tree-based algorithm for solving the problem [35].

### 5.4. Clique trees and the multifrontal method

Block algorithms have become increasingly important on advanced machine architectures, both in dense and sparse matrix computations [11]. The multifrontal factorization algorithm [8,32] is perhaps the canonical example in sparse matrix computation. That clique trees, which represent chordal graphs in block form, might be a useful tool in explaining the multifrontal method is not at all surprising.

Clique trees provide the framework for presenting the multifrontal algorithm in Peyton, Pothen, and Sun [34,38]. The clique tree is rooted and ordered by a postordering of the tree, and each clique $K$ has associated with it a frontal matrix $F(K)$. Let $K$ and $P$ be respectively a clique and its parent in the clique tree. The columns of $F(K)$ are partitioned into two sets: the factor columns of $F(K)$ correspond to the vertices in $K \setminus P$, and the update columns of $F(K)$ correspond to the vertices in $K \cap P$. For further details consult the two references given above.

Due to its simplicity, the *supernodal elimination tree* is more commonly used in descriptions of the multifrontal algorithm. Liu's survey article [32], for example, uses the supernodal elimination tree to describe the block version of the algorithm.

### 5.5. Future progress on the "ordering" problem

Finally, we anticipate that a solid understanding of chordal graphs and clique trees will play a role in future progress in the difficult area of analyzing and understanding ordering heuristics. The problem of finding a fill-minimizing ordering of an arbitrary graph is NP-hard [48]. Consequently, progress in understanding the "ordering" problem will probably require a better understanding of the broad but nontheless highly restricted classes of graphs $G_A$ that arise in various application areas. If there is some progress in that area, then we further speculate that creating and/or analyzing ordering algorithms for these classes of graphs will involve many interesting properties and features of chordal graphs and clique trees. Some will be the results presented in this paper; perhaps others will be new, or at least a fresh look at familiar concepts.

## 6. References

[1] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database systems. *J. Assoc. Comput. Mach.*, 30:479–513, 1983.

[2] P. A. Bernstein and N. Goodman. Power of natural semijoins. *SIAM J. Comput.*, 10:751–771, 1981.

[3] J.R.S. Blair, R.E. England, and M.G. Thomason. Cliques and their separators in triangulated graphs. Technical Report CS-78-88, Department of Computer Science, The University of Tennessee, Knoxville, Tennessee, 1988.

[4] J.R.S. Blair and B.W. Peyton. On finding minimum-diameter clique trees. Technical Report ORNL/TM-11850, Oak Ridge National Laboratory, Oak Ridge, TN, 1991.

[5] P. Buneman. A characterization of rigid circuit graphs. *Discrete Math.*, 9:205–212, 1974.

[6] G. A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg*, 25:71–76, 1961.

[7] I. S. Duff and J. K. Reid. A note on the work involved in no-fill sparse matrix factorization. *IMA J. Numer. Anal.*, 3:37–40, 1983.

[8] I.S. Duff and J.K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Software*, 9:302–325, 1983.

[9] P. Edelman and R.E. Jamison. The theory of convex geometries. *Geometriae Dedicata*, 19:247–270, 1985.

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

[11] K.A. Gallivan, M.T. Heath, E. Ng, J.M. Ortega, B.W. Peyton, R.J. Plemmons, C.H. Romine, A.H. Sameh, and R.G. Voigt. *Parallel Algorithms for Matrix Computations.* SIAM, Philadelphia, 1990.

[12] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combin. Theory Ser. B*, 16:47–56, 1974.

[13] F. Gavril. Generating the maximum spanning trees of a weighted graph. *J. Algorithms*, 8:592–597, 1987.

[14] A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10:345–363, 1973.

[15] A. George and J.W-H. Liu. An automatic nested dissection algorithm for irregular finite element problems. *SIAM J. Numer. Anal.*, 15:1053–1069, 1978.

[16] A. George and J.W-H. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.

[17] A. George and J.W-H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.

[18] A.M. Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, Cambridge, 1985.

[19] J.R. Gilbert and R. Schreiber. Highly parallel sparse Cholesky factorization. *SIAM J. Sci. Stat. Comput.*, 13:1151–1172, 1992.

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

[21] C-W. Ho and R. C. T. Lee. Counting clique trees and computing perfect elimination schemes in parallel. *Inform. Process. Lett.*, 31:61–68, 1989.

[22] J.A.G. Jess and H.G.M. Kees. A data structure for parallel L/U decomposition. *IEEE Trans. Comput.*, C-31:231–239, 1982.

[23] E.S. Kirsch. Practical parallel algorithms for chordal graphs. Master's thesis, Dept. of Computer Science, The University of Tennessee, 1989.

[24] J.G. Lewis, B.W. Peyton, and A. Pothen. A fast algorithm for reordering sparse matrices for parallel factorization. *SIAM J. Sci. Stat. Comput.*, 10:1156–1173, 1989.

[25] J. W-H. Liu and A. Mirzaian. A linear reordering algorithm for parallel pivoting of chordal graphs. *SIAM J. Disc. Math.*, 2:100–107, 1989.

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

[27] J.W-H. Liu. A compact row storage scheme for Cholesky factors using elimination trees. *ACM Trans. Math. Software*, 12:127–148, 1986.

[28] J.W-H. Liu. On the storage requirement in the out-of-core multifrontal method for sparse factorization. *ACM Trans. Math. Software*, 12:249–264, 1986.

[29] J.W-H. Liu. Equivalent sparse matrix reordering by elimination tree rotations. *SIAM J. Sci. Stat. Comput.*, 9:424–444, 1988.

[30] J.W-H. Liu. Reordering sparse matrices for parallel elimination. *Parallel Computing*, 11:73–91, 1989.

[31] J.W-H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11:134–172, 1990.

[32] J.W-H. Liu. The multifrontal method for sparse matrix solution: theory and practice. *SIAM Review*, 34:82–109, 1992.

[33] M.E. Lundquist. *Zero patterns, chordal graphs and matrix completions.* PhD thesis, Dept. of Mathematical Sciences, Clemson University, 1990.

[34] B.W. Peyton. *Some applications of clique trees to the solution of sparse linear systems.* PhD thesis, Dept. of Mathematical Sciences, Clemson University, 1986.

[35] B.W. Peyton, A. Pothen, and X. Yuan. A clique tree algorithm for partitioning chordal graphs for parallel sparse triangular solution. In preparation.

[36] B.W. Peyton, A. Pothen, and X. Yuan. Partitioning a chordal graph into transitive subgraphs for parallel sparse triangular solution. In preparation.

[37] A. Pothen and F.L. Alvarado. A fast reordering algorithm for parallel sparse triangular solution. *SIAM J. Sci. Stat. Comput.*, 13:645–653, 1992.

[38] A. Pothen and C. Sun. A distributed multifrontal algorithm using clique trees. Technical Report CS-91-24, Department of Computer Science, The Pennsylvania State University, University Park, PA, 1991.

[39] R.C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, pages 1389–1401, 1957.

[40] D.J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, 1972.

[41] D.J. Rose, R.E. Tarjan, and G.S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266–283, 1976.

[42] R. Schreiber. A new implementation of sparse Gaussian elimination. *ACM Trans. Math. Software*, 8:256–276, 1982.

[43] D.R. Shier. Some aspects of perfect elimination orderings in chordal graphs. *Discr. Appl. Math.*, 7:325–331, 1984.

[44] R.E. Tarjan. Maximum cardinality search and chordal graphs. Unpublished Lecture Notes CS 259, 1976.

[45] R.E. Tarjan. *Data Structures and Network Algorithms.* SIAM, Philadelphia, 1983.

[46] 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.*, 13:566–579, 1984.

[47] J.R. Walter. *Representations of rigid cycle graphs.* PhD thesis, Wayne State University, 1972.

[48] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.

## INTERNAL DISTRIBUTION

1. B.R. Appleton
2-3. T.S. Darland
4. E.F. D'Azevedo
5. J.M. Donato
6. J.J. Dongarra
7. G.A. Geist
8. M.R. Leuze
9. E.G. Ng
10. C.E. Oliver
11-15. B.W. Peyton
16-20. S.A. Raby
21. C.H. Romine
22. T.H. Rowan
23-27. R.F. Sincovec
28-32. R.C. Ward
33. P.H. Worley
34. Central Research Library
35. ORNL Patent Office
36. K-25 Appl Tech Library
37. Y-12 Technical Library
38. Lab Records Dept - RC
39-40. Laboratory Records Dept

## EXTERNAL DISTRIBUTION

41. Cleve Ashcraft, Boeing Computer Services, P.O. Box 24346, M/S 7L-21, Seattle, WA 98124-0346

42. Donald M. Austin, 6196 EECS Bldg., University of Minnesota, 200 Union St., S.E., Minneapolis, MN 55455

43. Robert G. Babb, Oregon Graduate Institute, CSE Department, 19600 N.W. von Neumann Drive, Beaverton, OR 97006-1999

44. Clive Baillie, Physics Department, Campus Box 390, University of Colorado, Boulder, CO 80309

45. Lawrence J. Baker, Exxon Production Research Company, P.O. Box 2189, Houston, TX 77252-2189

46. Jesse L. Barlow, Department of Computer Science, Pennsylvania State University, University Park, PA 16802

47. Edward H. Barsis, Computer Science and Mathematics, P.O. Box 5800, Sandia National Laboratories, Albuquerque, NM 87185

48. Chris Bischof, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

49. Ake Bjorck, Department of Mathematics, Linkoping University, S-581 83 Linkoping, Sweden

50-54. Jean R. S. Blair, Department of Computer Science, Ayres Hall, University of Tennessee, Knoxville, TN 37996-1301

55. Heather Booth, Department of Computer Science, 107 Ayres Hall, University of Tennessee, Knoxville, TN 37996-1301

56. Roger W. Brockett, Wang Professor of Electrical Engineering and Computer Science, Division of Applied Sciences, Harvard University, Cambridge, MA 02138

57. James C. Browne, Department of Computer Science, University of Texas, Austin, TX 78712

58. Bill L. Buzbee, Scientific Computing Division, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307

59. Donald A. Calahan, Department of Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI 48109

60. John Cavallini, Deputy Director, Scientific Computing Staff, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585

61. Ian Cavers, Department of Computer Science, University of British Columbia, Vancouver, British Columbia V6T 1W5, Canada

62. Tony Chan, Department of Mathematics, University of California, Los Angeles, 405 Hilgard Avenue, Los Angeles, CA 90024

63. Jagdish Chandra, Army Research Office, P.O. Box 12211, Research Triangle Park, NC 27709

64. Eleanor Chu, Department of Mathematics and Statistics, University of Guelph, Guelph, Ontario, Canada N1G 2W1

65. Melvyn Ciment, National Science Foundation, 1800 G Street N.W., Washington, DC 20550

66. Tom Coleman, Department of Computer Science, Cornell University, Ithaca, NY 14853

67. Paul Concus, Mathematics and Computing, Lawrence Berkeley Laboratory, Berkeley, CA 94720

68. Andy Conn, IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598

69. John M. Conroy, Supercomputer Research Center, 17100 Science Drive, Bowie, MD 20715-4300

70. Jane K. Cullum, IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598

71. George Cybenko, Center for Supercomputing Research and Development, University of Illinois, 104 S. Wright Street, Urbana, IL 61801-2932

72. George J. Davis, Department of Mathematics, Georgia State University, Atlanta, GA 30303

73. Tim A. Davis, Computer and Information Sciences Department, 301 CSE, University of Florida, Gainesville, FL 32611-2024

74. John J. Dorning, Department of Nuclear Engineering Physics, Thornton Hall, McCormick Road, University of Virginia, Charlottesville, VA 22901

75. Iain Duff, Atlas Centre, Rutherford Appleton Laboratory, Didcot, Oxon OX11 0QX, England

76. Patricia Eberlein, Department of Computer Science, SUNY at Buffalo, Buffalo, NY 14260

77. Stanley Eisenstat, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520

78. Lars Elden, Department of Mathematics, Linkoping University, 581 83 Linkoping, Sweden

79. Howard C. Elman, Computer Science Department, University of Maryland, College Park, MD 20742

80. Robert E. England, Mathematics and Computer Science Department, Northern Kentucky University, Highland Heights, KY 41076-1448

81. Albert M. Erisman, Boeing Computer Services, Engineering Technology Applications, ETA Division, P.O. Box 24346, MS-7L-20 Seattle, WA 98124-0346

82. Geoffrey C. Fox, Northeast Parallel Architectures Center, 111 College Place, Syracuse University, Syracuse, NY 13244-4100

83. Paul Frederickson, Los Alamos National Laboratory, Center for Research on Parallel Computing, MS B287, Los Alamos, NM 87545

84. Fred N. Fritsch, L-316, Computing and Mathematics Research Division, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550

85. Robert E. Funderlic, Department of Computer Science, North Carolina State University, Raleigh, NC 27650

86. K. Gallivan, Computer Science Department, University of Illinois, Urbana, IL 61801

87. Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47405

88. Feng Gao, Department of Computer Science, University of British Columbia, Vancouver, British Columbia V6T 1W5, Canada

89. David M. Gay, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

90. C. William Gear, NEC Research Institute, 4 Independence Way, Princeton, NJ 08540

91. W. Morven Gentleman, Division of Electrical Engineering, National Research Council, Building M-50, Room 344, Montreal Road, Ottawa, Ontario, Canada K1A 0R8

92. J. Alan George, Vice President, Academic and Provost, Needles Hall, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

93. John R. Gilbert, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto CA 94304

94. Gene H. Golub, Department of Computer Science, Stanford University, Stanford, CA 94305

95. Joseph F. Grcar, Division 8245, Sandia National Laboratories, Livermore, CA 94551-0969

96. John Gustafson, Ames Laboratory, Iowa State University, Ames, IA 50011

97. Per Christian Hansen, UCI*C Lyngby, Building 305, Technical University of Denmark, DK-2800 Lyngby, Denmark

98. Richard Hanson, IMSL Inc., 2500 Park West Tower One, 2500 City West Blvd., Houston, TX 77042-3020

99. Michael T. Heath, National Center for Supercomputing Applications, 4157 Beckman Institute, University of Illinois, 405 North Mathews Avenue, Urbana, IL 61801-2300

100. Stephen T. Hedetniemi, Department of Computer Science, Clemson University, Clemson, SC 29634

101. Don E. Heller, Physics and Computer Science Department, Shell Development Co., P.O. Box 481, Houston, TX 77001

102. Nicholas J. Higham, Department of Mathematics, University of Manchester, Grt Manchester, M13 9PL, England

103. Charles J. Holland, Air Force Office of Scientific Research, Building 410, Bolling Air Force Base, Washington, DC 20332

104. Robert E. Huddleston, Computation Department, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550

105. Ilse Ipsen, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520

106. Barry Joe, Department of Computer Science, University of Alberta, Edmonton, Alberta T6G 2H1, Canada

107. Lennart Johnsson, Thinking Machines Inc., 245 First Street, Cambridge, MA 02142-1214

108. Harry Jordan, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO 80309

109. Bo Kagstrom, Institute of Information Processing, University of Umea, 5-901 87 Umea, Sweden

110. Malvyn H. Kalos, Cornell Theory Center, Engineering and Theory Center Bldg., Cornell University, Ithaca, NY 14853-3901

111. Hans Kaper, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Bldg. 221, Argonne, IL 60439

112. Linda Kaufman, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

113. Robert J. Kee, Division 8245, Sandia National Laboratories, Livermore, CA 94551-0969

114. Kenneth Kennedy, Department of Computer Science, Rice University, P.O. Box 1892, Houston, TX 77001

115. Eric S. Kirsch, Department of Computer Science, 107 Ayres Hall, University of Tennessee, Knoxville, TN 37996-1301

116. Thomas Kitchens, Department of Energy, Scientific Computing Staff, Office of Energy Research, ER-7, Office G-236 Germantown, Washington, DC 20585

117. Michael A. Langston, Department of Computer Science, 107 Ayres Hall, University of Tennessee, Knoxville, TN 37996-1301

118. Richard Lau, Office of Naval Research, Code 111MA, 800 Quincy Street, Boston Tower 1, Arlington, VA 22217-5000

119. Alan J. Laub, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106

120. Robert L. Launer, Army Research Office, P.O. Box 12211, Research Triangle Park, NC 27709

121. Charles Lawson, MS 301-490, Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109

122. Peter D. Lax, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012

123. James E. Leiss, Rt. 2, Box 142C, Broadway, VA 22815

124. John G. Lewis, Boeing Computer Services, P.O. Box 24346, M/S 7L-21, Seattle, WA 98124-0346

125. Jing Li, IMSL Inc., 2500 Park West Tower One, 2500 City West Blvd., Houston, TX 77042-3020

126. Heather M. Liddell, Center for Parallel Computing, Department of Computer Science and Statistics, Queen Mary College, University of London, Mile End Road, London E1 4NS, England

127. Arno Liegmann, c/o ETH Rechenzentrum, Clausiusstr. 55, CH-8092 Zurich, Switzerland

128. Joseph Liu, Department of Computer Science, York University, 4700 Keele Street, North York, Ontario, Canada M3J 1P3

129. Robert F. Lucas, Supercomputer Research Center, 17100 Science Drive, Bowie, MD 20715-4300

130. Franklin Luk, Department of Computer Science, Amos Eaton Building - #131, Rensselaer Polytechnic Institute, Troy, NY 12180-3590

131. Brian A. Malloy, Department of Computer Science, Clemson University, Clemson, SC 29634

132. Thomas A. Manteuffel, Department of Mathematics, University of Colorado - Denver, Campus Box 170, P.O. Box 173364, Denver, CO 80217-3364

133. Consuelo Maulino, Universidad Central de Venezuela, Escuela de Computacion, Facultad de Ciencias, Apartado 47002, Caracas 1041-A, Venezuela

134. James McGraw, Lawrence Livermore National Laboratory, L-306, P.O. Box 808, Livermore, CA 94550

135. Paul C. Messina, Mail Code 158-79, California Institute of Technology, 1201 E. California Blvd., Pasadena, CA 91125

136. Cleve Moler, The Mathworks, 325 Linfield Place, Menlo Park, CA 94025

137. Neville Moray, Department of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green Street, Urbana, IL 61801

138. Dianne P. O'Leary, Computer Science Department, University of Maryland, College Park, MD 20742

139. James M. Ortega, Department of Applied Mathematics, Thornton Hall, University of Virginia, Charlottesville, VA 22901

140. Charles F. Osgood, National Security Agency, Ft. George G. Meade, MD 20755

141. Chris Paige, McGill University, School of Computer Science, McConnell Engineering Building, 3480 University Street, Montreal, Quebec, Canada H3A 2A7

142. Roy P. Pargas, Department of Computer Science, Clemson University, Clemson, SC 29634-1906

143. Beresford N. Parlett, Department of Mathematics, University of California, Berkeley, CA 94720

144. Merrell Patrick, Department of Computer Science, Duke University, Durham, NC 27706

145. Dan Pierce, Boeing Computer Services, P.O. Box 24346, M/S 7L-21 Seattle, WA 98124-0346

146. Robert J. Plemmons, Departments of Mathematics and Computer Science, Box 7311, Wake Forest University, Winston-Salem, NC 27109

147. Jesse Poore, Department of Computer Science, Ayres Hall, University of Tennessee, Knoxville, TN 37996-1301

148. Alex Pothen, Computer Science Department, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

149. Yuanchang Qi, IBM European Petroleum Application Center, P.O. Box 585, N-4040 Hafrsfjord, Norway

150. Giuseppe Radicati, IBM European Center for Scientific and Engineering Computing, via del Giorgione 159, I-00147 Roma, Italy

151. S. S. Ravi, Department of Computer Science, LI67A, 1400 Washington Avenue, Albany, NY 12222

152. John K. Reid, Numerical Analysis Group, Central Computing Department, Atlas Centre, Rutherford Appleton Laboratory, Didcot, Oxon OX11 0QX, England

153. Werner C. Rheinboldt, Department of Mathematics and Statistics, University of Pittsburgh, Pittsburgh, PA 15260

154. John R. Rice, Computer Science Department, Purdue University, West Lafayette, IN 47907

155. Garry Rodrigue, Numerical Mathematics Group, Lawrence Livermore Laboratory, Livermore, CA 94550

156. Donald J. Rose, Department of Computer Science, Duke University, Durham, NC 27706

157. Edward Rothberg, Department of Computer Science, Stanford University, Stanford, CA 94305

158. Axel Ruhe, Dept. of Computer Science, Chalmers University of Technology, S-41296 Goteborg, Sweden

159. Joel Saltz, ICASE, MS 132C, NASA Langley Research Center, Hampton, VA 23665

160. Ahmed H. Sameh, Center for Supercomputing R&D, 1384 W. Springfield Avenue, University of Illinois, Urbana, IL 61801

161. Michael Saunders, Systems Optimization Laboratory, Operations Research Department, Stanford University, Stanford, CA 94305

162. Robert Schreiber, RIACS, Mail Stop 230-5, NASA Ames Research Center, Moffet Field, CA 94035

163. Martin H. Schultz, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520

164. David S. Scott, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006

165. Lawrence F. Shampine, Mathematics Department, Southern Methodist University, Dallas, TX 75275

166. Andy Sherman, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520

167. Kermit Sigmon, Department of Mathematics, University of Florida, Gainesville, FL 32611

168. Horst Simon, Mail Stop T045-1, NASA Ames Research Center, Moffett Field, CA 94035

169. Anthony Skjellum, Lawrence Livermore National Laboratory, 7000 East Ave., L-316, P.O. Box 808 Livermore, CA 94551

170. Danny C. Sorensen, Department of Mathematical Sciences, Rice University, P.O. Box 1892, Houston, TX 77251

171. G. W. Stewart, Computer Science Department, University of Maryland, College Park, MD 20742

172. Paul N. Swartztrauber, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307

173. Michael G. Thomason, Department of Computer Science, 107 Ayres Hall, University of Tennessee, Knoxville, TN 37996-1301

174. Philippe Toint, Dept. of Mathematics, University of Namur, FUNOP, 61 rue de Bruxelles, B-Namur, Belgium

175. Bernard Tourancheau, LIP, ENS-Lyon, 69364 Lyon cedex 07, France

176. Henk van der Vorst, Dept. of Techn. Mathematics and Computer Science, Delft University of Technology, P.O. Box 356, NL-2600 AJ Delft, The Netherlands

177. Charles Van Loan, Department of Computer Science, Cornell University, Ithaca, NY 14853

178. Jim M. Varah, Centre for Integrated Computer Systems Research, University of British Columbia, Office 2053-2324 Main Mall, Vancouver, British Columbia V6T 1W5, Canada

179. Udaya B. Vemulapati, Dept. of Computer Science, University of Central Florida, Orlando, FL 32816-0362

180. Robert G. Voigt, ICASE, MS 132-C, NASA Langley Research Center, Hampton, VA 23665

181. Phuong Vu, Cray Research, Inc., 19607 Franz Rd., Houston, TX 77084

182. Daniel D. Warner, Department of Mathematical Sciences, O-104 Martin Hall, Clemson University, Clemson, SC 29631

183. Robert P. Weaver, 1555 Rockmont Circle, Boulder, CO 80303

184. Mary F. Wheeler, Rice University, Department of Mathematical Sciences, P.O. Box 1892, Houston, TX 77251

185. Andrew B. White, Computing Division, Los Alamos National Laboratory, P.O. Box 1663, MS-265, Los Alamos, NM 87545

186. Margaret Wright, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

187. David Young, University of Texas, Center for Numerical Analysis, RLM 13.150, Austin, TX 78731

188. Earl Zmijewski, Department of Computer Science, University of California, Santa Barbara, CA 93106

189. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P.O. Box 2001 Oak Ridge, TN 37831-8600

190–199. Office of Scientific & Technical Information, P.O. Box 62, Oak Ridge, TN 37831

DATE
FILMED
6/17/93