ORNL/TM-11850

Engineering Physics and Mathematics Division

Mathematical Sciences Section

ON FINDING MINIMUM-DIAMETER 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: December 1991

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
operated by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC-05-84OR21400

Contents

ON FINDING MINIMUM-DIAMETER CLIQUE TREES

Jean R. S. Blair
Barry W. Peyton

Abstract

It is well-known that any chordal graph can be represented as a clique tree (acyclic hypergraph, join tree). Since some chordal graphs have many distinct clique tree representations, it is interesting to consider which one is most desirable under various circumstances. A clique tree of minimum diameter (or height) is sometimes a natural candidate when choosing clique trees to be processed in a parallel computing environment.

This paper introduces a linear time algorithm for computing a minimum-diameter clique tree. The new algorithm is an analogue of the natural greedy algorithm for rooting an ordinary tree in order to minimize its height. It has potential application in the development of parallel algorithms for both knowledge-based systems and the solution of sparse linear systems of equations.

Key words. Chordal graph, clique tree, minimum spanning tree, minimum-diameter tree.

AMS(MOS) subject classifications. 68R10, 05C65, 65F50, 68Q25.

1. Introduction

Chordal graphs arise in several application areas including data-base management systems [1, 11, 30], knowledge-based systems [3, 19, 22], and the solution of sparse symmetric linear systems of equations [15, 23, 25, 26, 28]. A *clique tree* representation of a chordal graph often reduces the size of the data structure needed to store the graph, permitting the use of extremely efficient algorithms that take advantage of the compactness of the representation [22, 23, 30]. However, using a clique tree to represent a chordal graph is an ambiguous proposition in the sense that there may be more than one clique tree for a given chordal graph. In fact, Gavril, Ho, and Lee [14, 17] have shown that a tight upper bound on the number of distinct clique trees is an exponential function of the number of nodes in the graph. It is interesting from a theoretical point of view and potentially beneficial from a practical standpoint to consider how one clique tree representation may be better than another in a given context.

The algorithm presented in this paper is motivated primarily by the following question: Which clique trees are most suitable as input for *parallel* algorithms in various application areas? In at least some cases, a clique tree of minimum diameter (or, equivalently, minimum height)[1] is a natural candidate. In particular this is the case when the parallel algorithm in question has a leading (or otherwise significant) term in its time complexity that grows with the height of the clique tree. For the last two application areas mentioned above, we are aware of parallel algorithms under study for which this holds. Discussion of these application areas appears in the concluding section.

The essential character of the algorithm introduced here is very simple. Consider the problem of selecting a root that minimizes the height of a tree T. One way to solve this problem is a simple greedy algorithm that repeats the following major step until there are no nodes remaining in the tree: determine the leaf nodes (i.e., nodes of degree one) of the current tree and eliminate each of these nodes and the single edge incident upon it. The last major step eliminates either one or two nodes, and the height of T is minimized by rooting it at one of these nodes.

The algorithm presented here for finding a minimum-diameter clique tree is an analogue of this algorithm: it eliminates a large set of "leaf cliques" from the current chordal graph at each major step. One issue to be addressed is how this large set of leaf cliques can be computed with no *a priori* knowledge of a clique tree in which they are leaves. The first set of results deals with this issue. Subsequent results link the property of having a *maximum* number of leaves with reduction of the distances between cliques, after which it is quite easy to prove that the new algorithm works correctly.

Our algorithm is a greedy algorithm, and other greedy algorithms closely related

[1] This equivalence holds because a clique tree can be rooted at any node.

to ours have appeared in the literature [15, 25]. It should be noted that a minimum-diameter clique tree can be obtained easily from the output of an algorithm presented in Gilbert and Schreiber [15]. Moreover, the number of major steps in that algorithm is the height of a minimum-diameter clique tree. However, clique trees were not the object of study in [15], and furthermore the authors do not prove that the number of major steps is minimized. A simple extension of the analysis in this paper demonstrates that the algorithm in [15] does use the minimum number of major steps, and consequently has associated with it a minimum-diameter clique tree.

We hope the reader will find the lemmas and propositions leading up to our main result interesting in their own right. In selecting notation and organizing the material, we have striven to make the results easily accessible to as broad an audience as possible. More specifically, the results presented here may be of some value in providing a broad spectrum of readers with a more concrete grasp of the primary features and essential nature of clique trees and the chordal graphs they represent. In keeping with this expository goal, we have chosen to make most of the presentation as self-contained as possible; for example, we have included the proofs of two key results in the published literature.

The paper is organized as follows. Section 2 introduces some terminology and provides background results on clique trees. Section 3 contains a characterization of leaf cliques and also discusses clique trees that have as many leaves as possible. The new algorithm and its proof of correctness are found in Section 4. Lewis et al. [23] contains many of the details required to demonstrate that the new algorithm has a linear time implementation. Section 5 briefly outlines essential material from [23], presents a detailed version of our algorithm that addresses several key implementation issues not addressed in [23], and presents other material needed to verify the linear time complexity of the algorithm. Concluding remarks can be found in Section 6.

2. Clique trees: background

This section contains terminology and background material on clique trees. We assume the reader is familiar with standard graph terminology (see, for example, Golombic [16]). For easy reference we have included, in an appendix, a table of informal definitions for most of the notation introduced here and in later sections of the paper. To make the notation easier to read, we adhere to the following.

1. When needed, a subscript is used to identify which chordal graph or clique tree the item pertains to. This subscript is suppressed where the relevant graph is known by context.

2. In almost every case there is a strong mnemonic association between the symbol and what it represents: T for trees, K for cliques, etc.

2.1. Definition of clique trees

Let $G = (V, E)$ be a chordal[2] graph and $\boldsymbol{\mathcal{K}}_G = \{K_1, K_2, \ldots, K_m\}$ be the set containing the maximal cliques[3] of G. Throughout this paper, the graph G is assumed to be connected merely to avoid vacuous technical clutter in the proofs. That all definitions and results contained in this paper generalize immediately to disconnected chordal graphs should be readily apparent.

While many characterizations and properties of chordal graphs have appeared in the published literature [1, 6, 12, 16, 28], we restrict our attention to those results connecting chordal graphs to *clique trees* (also called acyclic hypergraphs or join trees). Our departure point in defining clique trees is the following well-known result, which we state without proof.

Theorem 2.1 (Walter [31], Gavril [13], Buneman [4]). *A graph $G = (V, E)$ is chordal if and only if there exists a tree $T = (\boldsymbol{\mathcal{K}}_G, \boldsymbol{\mathcal{E}})$ that satisfies the following property: for every node $v \in V$, the set of cliques containing v induces a subtree of T.*

For any chordal graph G, we shall let $\boldsymbol{\mathcal{T}}_G$ denote the set of all trees $T = (\boldsymbol{\mathcal{K}}_G, \boldsymbol{\mathcal{E}})$ that satisfy this property, and we shall refer to any member of $\boldsymbol{\mathcal{T}}_G$ as a *clique tree* of the underlying chordal graph G.

2.2. Maximum spanning tree characterization

Associated with each chordal graph G is a *clique intersection graph* defined as follows. The node set of the clique intersection graph is the set of cliques $\boldsymbol{\mathcal{K}}_G$. Two distinct cliques K and K' are connected by an edge if their intersection is nonempty; moreover, each such edge $\{K, K'\}$ is assigned a positive weight given by $|K \cap K'|$.

Bernstein, Goodman, and Gavril [2, 14] have shown that the set of clique trees $\boldsymbol{\mathcal{T}}_G$ is precisely the set of maximum-weight spanning trees of the clique intersection graph associated with G. The proof of this result, included here in Theorem 2.2, closely follows the one found in Gavril [14]. To prove Theorem 2.2 and later results we need the following simple corollary of Theorem 2.1.

Corollary 1. *A tree $T = (\boldsymbol{\mathcal{K}}_G, \boldsymbol{\mathcal{E}})$ is a clique tree of G if and only if for every pair of distinct cliques $K, K' \in \boldsymbol{\mathcal{K}}_G$, the intersection $K \cap K'$ is contained in every clique on the path connecting K and K' in the tree.*

Proof. The result follows easily from the definition of clique trees and the fact that the intersection of any two subtrees of a tree is likewise a tree. ■

---

[2]A graph is *chordal* (triangulated, rigid circuit) if every cycle of length ≥ 4 contains a *chord*, i.e., an edge connecting two non-adjacent nodes in the cycle.

[3]Throughout this paper the term *clique* always refers to a maximal clique. The term *maximal clique* is used only where emphasis on maximality seems warranted. To avoid confusion, any *submaximal clique* is referred to as a *complete subgraph* of G.

Theorem 2.2 (Bernstein [2]). *A tree $T = (\boldsymbol{\mathcal{K}}_G, \boldsymbol{\mathcal{E}})$ is a clique tree of G if and only if it is a maximum-weight spanning tree of the clique intersection graph of G.*

Proof. First, assume T is a clique tree and choose two cliques K and K' not connected by an edge in T. Consider the cycle formed by adding the edge $\{K, K'\}$ to T. By Corollary 1, every edge along this path has weight no smaller than $|K \cap K'|$. It is well-known that a tree with this property is a maximum-weight spanning tree of the graph (see, for example, [29, pp. 71–72]).

Now, suppose that T_{mst} is a maximum-weighted spanning tree of the clique intersection graph of G. Let T_{ct} be a clique tree having a maximum number of edges in common with T_{mst}. Assume for the purpose of contradiction that there is an edge $\{K_1, K_2\}$ of T_{mst} that is not an edge of T_{ct}. Let $T_1 = (\boldsymbol{\mathcal{K}}_1, \boldsymbol{\mathcal{E}}_1)$ and $T_2 = (\boldsymbol{\mathcal{K}}_2, \boldsymbol{\mathcal{E}}_2)$ be the two subtrees of T_{mst} obtained by removing the edge $\{K_1, K_2\}$ from T_{mst}. Note that $\{\boldsymbol{\mathcal{K}}_1, \boldsymbol{\mathcal{K}}_2\}$ partitions $\boldsymbol{\mathcal{K}}_G$, and associated with this partition is a *cut set* of edges consisting of all edges in the clique interection graph with one end-point in $\boldsymbol{\mathcal{K}}_1$ and the other in $\boldsymbol{\mathcal{K}}_2$. It is well-known that any cycle in the clique intersection graph containing one edge from the cut set must contain another edge from the cut set as well. Now, consider the cycle (in T_{ct}) obtained by adding the edge $\{K_1, K_2\}$ to T_{ct}, and select from this cycle in T_{ct} one of the edges $\{K_3, K_4\} \neq \{K_1, K_2\}$ that belongs to the cut set.

Note that $\{K_3, K_4\}$ 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 Corollary 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 $\{K_1, K_2\}$. We leave it for the reader to verify that the resulting tree satisfies Theorem 2.1, and is thus a clique tree of G. This 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. ∎

2.3. Clique tree edges and graph separators

A node separator $S \subset V$ for two nodes a and b is any node set whose removal from G results in a graph in which a and b are in separate connected components. If no proper subset of S has this property, then S is said to be a *minimal a-b separator.* A well-known result states that a graph G is chordal if and only if every minimal a-b separator is a complete subgraph of G [6, 16, 28]. For any clique tree $T = (\boldsymbol{\mathcal{K}}_G, \boldsymbol{\mathcal{E}}) \in \boldsymbol{\mathcal{T}}_G$ consider the *multiset* defined by

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

Ho and Lee [18] showed that for each minimal *a*-*b* separator S of G, $\boldsymbol{\mathcal{M}}_T$ contains a number of copies of S that does not vary with T. (For brevity, we will refer to each member of $\boldsymbol{\mathcal{M}}_T$ as a *separator*.) Below, we provide a simple proof of the invariance of $\boldsymbol{\mathcal{M}}_T$ over all clique trees $T \in \boldsymbol{\mathcal{T}}_G$, which can be viewed as a weaker form of the result in [18].

Theorem 2.3 (Ho and Lee [18]). *The multiset of separators is the same for every clique tree $T \in \boldsymbol{\mathcal{T}}_G$.*

Proof. For the purpose of contradiction, suppose there exist two distinct clique trees $T, T' \in \boldsymbol{\mathcal{T}}_G$ for which $\boldsymbol{\mathcal{M}}_T \neq \boldsymbol{\mathcal{M}}_{T'}$. From among the clique trees $T' \in \boldsymbol{\mathcal{T}}_G$ for which $\boldsymbol{\mathcal{M}}_{T'} \neq \boldsymbol{\mathcal{M}}_T$, 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'. Consider the cycle obtained by adding the edge $\{K_1, K_2\}$ to T'. There must be an edge $\{K_3, K_4\}$ of the cycle that is contained in T' but not in T. From Corollary 1 and Theorem 2.2 it follows that $K_3 \cap K_4 = K_1 \cap K_2$. By Theorem 2.2, replacing $\{K_3, K_4\}$ in T' with $\{K_1, K_2\}$ results in a clique tree, and moreover the multiset of separators is the same as that of T'. Since the modified tree shares one more edge with T, contrary to our assumption about T', the result follows. ■

Henceforth, let $\boldsymbol{\mathcal{M}}_G$ denote the *multiset of separators* associated with each clique tree in $\boldsymbol{\mathcal{T}}_G$. For any set of nodes $S \subseteq V$, the *set of cliques containing* S, denoted by $\boldsymbol{\mathcal{K}}(S)$, is given by

$$\boldsymbol{\mathcal{K}}(S) := \{K \in \boldsymbol{\mathcal{K}}_G \mid S \subseteq K\}.$$

(Usually S will be a separator taken from $\boldsymbol{\mathcal{M}}_G$.) It is worth emphasizing that every separator $S \in \boldsymbol{\mathcal{M}}_G$ is contained in at least two cliques (i.e., $|\boldsymbol{\mathcal{K}}(S)| \geq 2$).

For any clique K, the *set of separators belonging to* K, denoted by $\boldsymbol{\mathcal{S}}(K)$, is given by

$$\boldsymbol{\mathcal{S}}(K) := \{S \in \boldsymbol{\mathcal{M}}_G \mid S \subset K\}.$$

Note that $\boldsymbol{\mathcal{S}}(K)$ contains *one copy* of each member of the multiset $\boldsymbol{\mathcal{M}}_G$ that is contained in K. The set $\overline{\boldsymbol{\mathcal{S}}}(K)$ contains each separator from $\boldsymbol{\mathcal{S}}(K)$ that is maximal with respect to set inclusion among the members of $\boldsymbol{\mathcal{S}}(K)$. In other words, $\overline{\boldsymbol{\mathcal{S}}}(K)$ is given by

$$\overline{\boldsymbol{\mathcal{S}}}(K) := \{S \in \boldsymbol{\mathcal{S}}(K) \mid S \text{ is properly contained in no separator } S' \in \boldsymbol{\mathcal{S}}(K)\}.$$

Loosely speaking, the following simple lemma states that in any clique tree the members of $\overline{\boldsymbol{\mathcal{S}}}(K)$ must be "used" by at least one of the tree edges incident on K.

Lemma 1. *Let $K \in \boldsymbol{\mathcal{K}}_G$ and $T \in \boldsymbol{\mathcal{T}}_G$. Then for every separator $S \in \overline{\boldsymbol{\mathcal{S}}}(K)$ there is at least one edge $\{K, K'\}$ of T for which $K \cap K' = S$.*

Proof. Choose a separator $S \in \overline{\boldsymbol{\mathcal{S}}}(K)$, and choose $P \in \boldsymbol{\mathcal{K}}(S) - \{K\}$.[4] Consider the path $K = K_1, K_2, \ldots, K_r = P$ from K to P in T. It follows from Corollary 1 that $S \subseteq K_i$ for $1 \leq i \leq r$, and hence $S \subseteq K \cap K_2$. From the maximality of S among the separators in $\boldsymbol{\mathcal{S}}(K)$ we have $K \cap K_2 = S$, which proves the result. ∎

3. Leaf cliques

For any clique tree $T \in \boldsymbol{\mathcal{T}}_G$, let $\boldsymbol{\mathcal{L}}_T$ $(\subseteq \boldsymbol{\mathcal{K}}_G)$ be the set containing the leaves of T (i.e, the members of $\boldsymbol{\mathcal{K}}_G$ with degree one in T). We then let $\boldsymbol{\mathcal{L}}_G$, the *leaf cliques of* G, be the set containing every clique that is a leaf in at least one clique tree $T \in \boldsymbol{\mathcal{T}}_G$. Practical implementation of our minimum-diameter clique tree algorithm requires access to the set $\boldsymbol{\mathcal{L}}_G$. The first subsection below contains a simple characterization of $\boldsymbol{\mathcal{L}}_G$ that ultimately leads to an efficient method for computing $\boldsymbol{\mathcal{L}}_G$. With $\boldsymbol{\mathcal{L}}_G$ in hand, the minimum-diameter clique tree algorithm must then compute a set of leaf cliques $\boldsymbol{\mathcal{L}}_{max} \subseteq \boldsymbol{\mathcal{L}}_G$ such that $\boldsymbol{\mathcal{L}}_{max} = \boldsymbol{\mathcal{L}}_T$ for at least one clique tree $T \in \boldsymbol{\mathcal{T}}_G$, and moreover $|\boldsymbol{\mathcal{L}}_T| \geq |\boldsymbol{\mathcal{L}}_{T'}|$ for every clique tree $T' \in \boldsymbol{\mathcal{T}}_G$. The second subsection contains a characterization of these *maximum cardinality leaf sets* $\boldsymbol{\mathcal{L}}_{max} \subseteq \boldsymbol{\mathcal{L}}_G$. An efficient method for computing $\boldsymbol{\mathcal{L}}_{max}$ is presented later in Section 5.1.

3.1. A characterization of leaf cliques

The next lemma gives a sufficient condition for membership in $\boldsymbol{\mathcal{L}}_G$. The proof of this lemma and the specific clique tree T' constructed in the proof are important departure points in the next section. Lemma 3 confirms that the condition in Lemma 2 is necessary as well as sufficient.

Lemma 2. *If $|\overline{\boldsymbol{\mathcal{S}}}(K)| = 1$, then K is a leaf in some clique tree $T' \in \boldsymbol{\mathcal{T}}_G$.*

Proof. Let S be the sole member of $\overline{\boldsymbol{\mathcal{S}}}(K)$, and suppose that K is *not* a leaf of $T \in \boldsymbol{\mathcal{T}}_G$ (see Figure 3.1). Choose $P \in \boldsymbol{\mathcal{K}}(S) - \{K\}$. It follows from Corollary 1 that $S \subseteq P'$ where P' is the clique adjacent to K on the path from K to P in T (possibly $P' = P$). Consider a clique $C_1 \neq P'$ that is also adjacent to K in T. (Such a clique must exist since K is not a leaf in T.) By Corollary 1, $C_1 \cap P \subseteq C_1 \cap K$. Furthermore, since S is the only member of $\overline{\boldsymbol{\mathcal{S}}}(K)$, we have $C_1 \cap K \subseteq S \subset P$. It follows that $C_1 \cap K = C_1 \cap P$, and hence the edges $\{C_1, K\}$ and $\{C_1, P\}$ have the same weight. Thus, by Theorem 2.2 the tree obtained from T by first removing the edge $\{C_1, K\}$ and then adding the edge $\{C_1, P\}$ is also a clique tree. Repeating this process for every clique $C_i \neq P'$ adjacent to K in T, we obtain a new clique tree T' in which K is a leaf (see Figure 3.1), and

---

[4] Throughout this paper the binary set difference operator is "$-$".

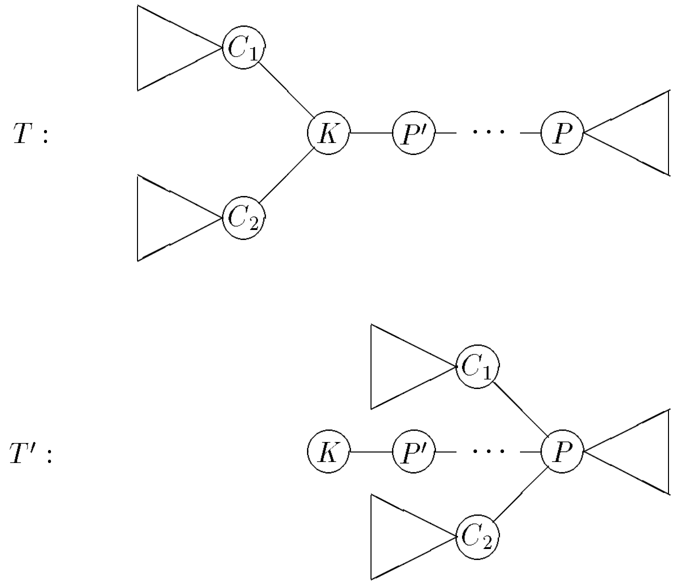

Figure 3.1: Transformation of T into T' in which K is a leaf, as discussed in the proof of Lemma 2.

this concludes the proof. ■

The specific operation that transformed the clique tree T (in which K is not a leaf) into the clique tree T' (in which K is a leaf) will be used in several subsequent proofs. We note here that the parameters required for this operation are a clique tree T, a leaf clique $K \in \boldsymbol{\mathcal{L}}_G - \boldsymbol{\mathcal{L}}_T$ and an arbitrary clique $P \neq K$ for which $\overline{\boldsymbol{\mathcal{S}}}(K) = \{K \cap P\}$. When P is not adjacent to K in T, these two cliques determine a third clique of interest, namely the clique P' adjacent to K on the path in T connecting K and P. Since $K \cap P' = K \cap P$, P' can play the role of P, as will be the case in an important application of this operation in Section 4. However, when this operation is used in other proofs, P will be chosen in such a way that it may not be adjacent to K.

The next lemma completes the first characterization of the cliques in $\boldsymbol{\mathcal{L}}_G$.

Lemma 3. *$K \in \boldsymbol{\mathcal{L}}_G$ if and only if* $|\overline{\boldsymbol{\mathcal{S}}}(K)| = 1$.

Proof. Sufficiency follows immediately from Lemma 2. To prove necessity, choose $K \in \boldsymbol{\mathcal{L}}_G$ and let $T \in \boldsymbol{\mathcal{T}}_G$ be a clique tree in which K is a leaf. Let P' be the single clique adjacent to K in T. Since $K \cap P'$ is the only separator associated with an edge incident on K in T, it follows from Lemma 1 that $K \cap P'$ is the only member of $\overline{\boldsymbol{\mathcal{S}}}(K)$. ■

A node in an ordinary tree is a leaf if it has only one neighbor. Lemma 3 is an analogue of this property for leaf cliques of a chordal graph. That is, a clique in a chordal graph is a leaf clique if it has only one *maximal* separator through which it can be joined to neighbors in a clique tree.

3.2. Maximum cardinality leaf sets

It is interesting to consider precisely which subsets of $\boldsymbol{\mathcal{L}}_G$ constitute a set of leaves $\boldsymbol{\mathcal{L}}_T$ for at least one clique tree $T \in \boldsymbol{\mathcal{T}}_G$. However, for our purposes we restrict our attention to a simpler question, that of characterizing the leaf sets $\boldsymbol{\mathcal{L}}_T$ of maximum cardinality. That is, we need a useful characterization of the leaf sets $\boldsymbol{\mathcal{L}}_T$ for which $|\boldsymbol{\mathcal{L}}_T| \geq |\boldsymbol{\mathcal{L}}_{T'}|$ for every clique tree $T' \in \boldsymbol{\mathcal{T}}_G$.

To that end, we introduce some more notation. We have shown in Lemma 3 that each leaf clique $K \in \boldsymbol{\mathcal{L}}_G$ has associated with it a single separator $S \in \boldsymbol{\mathcal{M}}_G$ that is maximal among the separators contained in K. Let $\overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_G)$ be the set containing a single copy of each separator associated with a leaf in this manner. More precisely,

$$\overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_G) := \{S \in \boldsymbol{\mathcal{M}}_G \mid \overline{\boldsymbol{\mathcal{S}}}(K) = \{S\} \text{ for some } K \in \boldsymbol{\mathcal{L}}_G\}.$$

For every *leaf separator* $S \in \overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_G)$, let $\boldsymbol{\mathcal{L}}(S)$ be the subset of $\boldsymbol{\mathcal{L}}_G$ defined by

$$\boldsymbol{\mathcal{L}}(S) := \{K \in \boldsymbol{\mathcal{L}}_G \mid \overline{\boldsymbol{\mathcal{S}}}(K) = \{S\}\}.$$

More informally, $\boldsymbol{\mathcal{L}}(S)$ contains the "cohort" of leaf cliques clustered around the leaf separator S. It is important to note that $\boldsymbol{\mathcal{L}}(S)$ may be a proper subset of the set of leaf cliques that contain S. For two leaf separators $S, S' \in \overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_G)$ where $S \subset S'$, any clique $K \in \boldsymbol{\mathcal{L}}(S')$ contains both leaf separators S and S'. In this case, however, we observe that $K \notin \boldsymbol{\mathcal{L}}(S)$ even though $K \in \boldsymbol{\mathcal{K}}(S)$. Indeed, each leaf belongs to precisely one leaf cohort set, and therefore the collection of leaf cohort sets

$$\{\boldsymbol{\mathcal{L}}(S) \mid S \in \overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_G)\}$$

forms a partition of $\boldsymbol{\mathcal{L}}_G$.

Lemma 4. *Assume $|\boldsymbol{\mathcal{K}}_G| \geq 3$. For a leaf separator $S \in \overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_G)$, there exists a clique tree $T \in \boldsymbol{\mathcal{T}}_G$ for which $\boldsymbol{\mathcal{L}}(S) \subseteq \boldsymbol{\mathcal{L}}_T$ if and only if $\boldsymbol{\mathcal{L}}(S) \subset \boldsymbol{\mathcal{K}}(S)$ (i.e., $\boldsymbol{\mathcal{K}}(S) - \boldsymbol{\mathcal{L}}(S) \neq \emptyset$).*

Proof. Choose a leaf separator $S \in \overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_G)$ and assume that $\boldsymbol{\mathcal{L}}(S) \subseteq \boldsymbol{\mathcal{L}}_T$ for some clique tree $T \in \boldsymbol{\mathcal{T}}_G$. It follows from Corollary 1 that $\boldsymbol{\mathcal{K}}(S)$ induces a subtree of T. Since $|\boldsymbol{\mathcal{K}}_G| \geq 3$ and $|\boldsymbol{\mathcal{K}}(S)| \geq 2$, $\boldsymbol{\mathcal{K}}(S)$ contains an interior clique P of T (i.e., $P \in \boldsymbol{\mathcal{K}}(S) - \boldsymbol{\mathcal{L}}_T$). Since $\boldsymbol{\mathcal{L}}(S) \subseteq \boldsymbol{\mathcal{L}}_T$, we have $P \in \boldsymbol{\mathcal{K}}(S) - \boldsymbol{\mathcal{L}}(S)$, completing the first half of the proof.

To prove the converse assume $\boldsymbol{\mathcal{K}}(S) - \boldsymbol{\mathcal{L}}(S) \neq \emptyset$, and let $P \in \boldsymbol{\mathcal{K}}(S) - \boldsymbol{\mathcal{L}}(S)$. Let $T \in \boldsymbol{\mathcal{T}}_G$, and suppose that there exists a clique $K \in \boldsymbol{\mathcal{L}}(S) - \boldsymbol{\mathcal{L}}_T$. As in the proof of Lemma 2 (see Figure 3.1), we can replace each edge $\{C, K\}$ incident on K (except $\{K, P'\}$) with the corresponding edge $\{C, P\}$ to obtain a clique tree T' in which K is a leaf. Repeating this operation for each clique in $\mathcal{L}(S) - \mathcal{L}_T$ transforms T into a clique tree T' for which $\mathcal{L}(S) \subseteq \mathcal{L}_{T'}$, giving us the result. ■

Lemma 5. *Assume $|\mathcal{K}_G| \geq 3$. Then $|\mathcal{L}_T| \geq |\mathcal{L}_{T'}|$ for every $T' \in \mathcal{T}_G$ if and only if T has the following two properties:*

1. *For $S \in \overline{\mathcal{S}}(\mathcal{L}_G)$, if $\mathcal{L}(S) \subset \mathcal{K}(S)$, then $\mathcal{L}(S) \subseteq \mathcal{L}_T$.*
2. *For $S \in \overline{\mathcal{S}}(\mathcal{L}_G)$, if $\mathcal{L}(S) = \mathcal{K}(S)$, then $\mathcal{L}_T$ contains all but one of the cliques in $\mathcal{L}(S)$.*

Proof. Suppose properties 1 and 2 hold. By Lemma 4, if $\mathcal{L}(S) = \mathcal{K}(S)$, then for every clique tree $T \in \mathcal{T}_G$ at least one member of $\mathcal{L}(S)$ is excluded from $\mathcal{L}_T$. It follows that no clique tree can have more leaves than one that posesses properties 1 and 2.

Suppose property 1 does not hold for some clique tree $T \in \mathcal{T}_G$. Then for some leaf separator $S \in \overline{\mathcal{S}}(\mathcal{L}_G)$ for which $\mathcal{L}(S) \subset \mathcal{K}(S)$ we have a clique $K \in \mathcal{L}(S)$ that is not a leaf of T. Since $|\mathcal{K}_G| \geq 3$ and $|\mathcal{K}(S)| \geq 2$, we can choose an interior clique $P \in \mathcal{K}(S) - \mathcal{L}_T$. As in the proof of Lemma 2 (see Figure 3.1), we can replace each edge $\{C, K\}$ incident on K (except $\{K, P'\}$) with the corresponding edge $\{C, P\}$ to obtain a clique tree T' in which K is a leaf. Note that P is the only clique in T' with more neighbors than it had in T. Since P is not a leaf in T and all leaves of T remain leaves in T', it follows that T' has one more leaf than T, and hence property 1 holds for any clique tree that has the maximum number of leaves.

A similar argument can be used to verify property 2, as follows. Suppose property 2 does not hold for some clique tree $T \in \mathcal{T}_G$. Then $|\mathcal{L}(S) - \mathcal{L}_T| \neq 1$ for some leaf separator $S \in \overline{\mathcal{S}}(\mathcal{L}_G)$ for which $\mathcal{L}(S) = \mathcal{K}(S)$. From Lemma 4 we know that $|\mathcal{L}(S) - \mathcal{L}_T| \neq 0$. Thus, we have two or more cliques in $\mathcal{L}(S)$ that are not leaves of T. Let K and P be two such cliques. From this point, the argument runs the same course as that found in the previous paragraph, completing the proof. ■

Lemma 5 characterizes maximum cardinality leaf sets $\mathcal{L}_{max}$: for each leaf separator $S \in \overline{\mathcal{S}}(\mathcal{L}_G)$, either all of the leaf cliques from $\mathcal{L}(S)$ are included in $\mathcal{L}_{max}$, or all but one of the leaf cliques from $\mathcal{L}(S)$ are included in $\mathcal{L}_{max}$. Furthermore, exclusion of a single clique of $\mathcal{L}(S)$ from $\mathcal{L}_{max}$ occurs if and only if the only cliques containing S are those in $\mathcal{L}(S)$ (i.e., $\mathcal{K}(S) = \mathcal{L}(S)$).

4. A minimum-diameter clique tree algorithm

The characterization of maximum cardinality leaf sets given in the previous section provides the basis for an algorithm that generates minimum-diameter clique trees from the bottom up (i.e., from the leaves up to the root). This section gives a high-level description of our bottom-up algorithm and proves that it does indeed generate a mimimum-diameter clique tree.

Algorithms for generating clique trees are based on the detection of simplicial nodes, which are defined as follows. A *simplicial node* in G is any node whose adjacency set forms a complete subgraph in G. It is well-known that any non-trivial chordal graph has at least two simplicial nodes [6], and moreover that any chordal graph can be reduced to the null graph by successive removal of simplical nodes [6, 12, 16, 28]. The order in which the simplicial nodes are removed is known as a *perfect elimination ordering.* It is trivial to show that a node is simplicial if and only if it belongs to precisely one maximal clique. This simple characterization of simplicial nodes has been useful in various applications. The result is stated formally and proved in [21, 23], and it is stated informally and used without proof in [8, 15, 30].

Maximum cardinality search [30] is a linear-time algorithm for generating a perfect elimination ordering of a chordal graph. With a few simple extensions, this algorithm can also generate the set of cliques $\boldsymbol{\mathcal{K}}_G$ and the set of edges $\boldsymbol{\mathcal{E}}$ of a clique tree $T \in \boldsymbol{\mathcal{T}}_G$ [3, 26, 30]. Blair et al. [3] have shown that this algorithm for generating a clique tree can be viewed as Prim's algorithm [27] for finding a maximum-weight spanning tree of the clique intersection graph of G. The algorithm presented in this paper however does not fall within this general framework, where the clique tree is generated from a known root in top-down fashion to the leaves. Instead, the new algorithm must generate a clique tree from an initial set of leaves in a bottom-up fashion, eventually determining a root clique that is *not known in advance*.

The next subsection describes an algorithm for generating an arbitrary clique tree from the bottom up by successive removal of leaf cliques from the "current" chordal graph. This approach to generating clique trees is refined in Section 4.2 where we present our new algorithm, which removes a maximum cardinality leaf set at each major step. Our main result, presented in Section 4.3, demonstrates that a clique tree generated by the new algorithm also has minimum diameter.

4.1. A bottom-up clique tree algorithm

To describe the bottom-up algorithm for generating an arbitrary clique tree, we need the following definitions and notation. For each clique $K \in \boldsymbol{\mathcal{K}}_G$ we define the parameter $\rho(K)$ to be the number of simplicial nodes contained in K. Another needed parameter is $\sigma(K)$, defined to be the size of the largest separator in $\overline{\boldsymbol{\mathcal{S}}}(K)$. Lemma 6 contains a formula for $\sigma(K)$ that is useful in Section 5. Lemma 7 uses the parameters $\sigma(K)$ and $\rho(K)$ to characterize $\boldsymbol{\mathcal{L}}_G$.

Lemma 6. *For* $K \in \boldsymbol{\mathcal{K}}_G$,

$$\sigma(K) = \max \{ \ |K \cap K'| \text{ for all } K' \in \boldsymbol{\mathcal{K}}_G - \{K\} \ \}.$$

Proof. Let $K, K' \in \boldsymbol{\mathcal{K}}_G$. Applying Corollary 1 to any clique tree $T \in \boldsymbol{\mathcal{T}}_G$, we have $K \cap K' \subseteq S$ for some separator $S \in \boldsymbol{\mathcal{S}}(K)$. The definition of $\sigma(K)$ implies that $\sigma(K) \geq |K \cap K'|$, and implies moreover that there exists $K'' \in \boldsymbol{\mathcal{K}}_G - \{K\}$ such that $\sigma(K) = |K \cap K''|$, which proves the result. ■

Lemma 7. *$K \in \boldsymbol{\mathcal{L}}_G$ if and only if $\sigma(K) + \rho(K) = |K|$.*

Proof. Suppose $K \in \boldsymbol{\mathcal{L}}_G$ and let S be the single separator in $\overline{\boldsymbol{\mathcal{S}}}(K)$ (see Lemma 3). From Corollary 1 and the fact that $\overline{\boldsymbol{\mathcal{S}}}(K) = \{S\}$, we have $K \cap K' \subseteq S$ for every clique $K' \in \boldsymbol{\mathcal{K}}_G - \{K\}$. Hence, any node $v \in K - S$ is found in no other clique of the graph. Thus, we have $\rho(K) \geq |K - S|$, and since $|\boldsymbol{\mathcal{K}}(S)| \geq 2$, none of the nodes in S is simplicial. Therefore $\rho(K) = |K - S|$, which along with $\sigma(K) = |S|$, proves necessity.

To prove sufficiency, suppose $\sigma(K) + \rho(K) = |K|$. Choose $S \in \overline{\boldsymbol{\mathcal{S}}}(K)$ such that $|S| = \sigma(K)$. Since $\rho(K) = |K| - \sigma(K)$ and none of the nodes in S are simplicial, every node in $K - S$ is simplicial. Since each simplicial node belongs to only one clique, it can belong to no separator in $\boldsymbol{\mathcal{M}}_G$. Consequently, $\overline{\boldsymbol{\mathcal{S}}}(K) = \{S\}$, and by Lemma 3, $K \in \boldsymbol{\mathcal{L}}_G$. ■

From Lemma 7, it is clear that any leaf clique $K \in \boldsymbol{\mathcal{L}}_G$ can be partitioned into two sets

$$K = Sim(K) \cup Sep(K),$$

where $Sim(K)$ contains $\rho(K)$ simplicial nodes and $Sep(K)$ contains the $\sigma(K)$ nodes that constitute the leaf separator associated with K.

Our bottom-up algorithm for generating a clique tree is based on successive elimination of leaf cliques from the chordal graph. For $K \in \boldsymbol{\mathcal{L}}_G$, $G \setminus Sim(K)$ denotes the graph obtained by eliminating the simplicial nodes in $Sim(K)$ and their incident edges from G. In other words, $G \setminus Sim(K)$ is the subgraph of G induced by $V - Sim(K)$. Since $Sep(K)$, which contains the nodes of K remaining in $G \setminus Sim(K)$, is contained in at least one other maximal clique K' of G, K disappears from $G \setminus Sim(K)$ as a maximal clique, and can be viewed as “absorbed” by K'. Moreover, since the nodes of $Sim(K)$ belong to no other maximal clique of G, the other maximal cliques of G remain unchanged in $G \setminus Sim(K)$. Thus, $G \setminus Sim(K)$ is precisely the chordal graph whose set of maximal cliques is given by $\boldsymbol{\mathcal{K}}_G - \{K\}$.

Figure 4.1 displays an algorithm for generating a clique tree from the bottom up, and Lemma 8 confirms that the algorithm is correct. Note that the algorithm generates a sequence of chordal graphs. From now on a subscript is used, as needed, to identify which graph a set or parameter pertains to (e.g., $Sim_H(K)$ in line 7 of the algorithm).

Lemma 8. *The algorithm in Figure 4.1 generates a clique tree.*

Proof. It is easy to show that the edge set contained in $\boldsymbol{\mathcal{E}}$ at the end of the algorithm is the edge set of a tree T, and we leave it for the reader to verify this. The proof

01 $\boldsymbol{\mathcal{E}} \leftarrow \emptyset$
02 $H \leftarrow G$
03 **while** $|\boldsymbol{\mathcal{K}}_H| \geq 2$ **do**
04   Choose $K \in \boldsymbol{\mathcal{L}}_H$
05   Choose $P \in \boldsymbol{\mathcal{K}}_H - \{K\}$ for which $Sep_H(K) \subset P$
06     $\boldsymbol{\mathcal{E}} \leftarrow \boldsymbol{\mathcal{E}} \cup \{K, P\}$
07     $H \leftarrow H \setminus Sim_H(K)$
08 **end while**

Figure 4.1: Algorithm for generating a clique tree.

that T is indeed a clique tree is by induction on the number of cliques in the graph. The base step is trivial. For the induction step, let G be a chordal graph with $m \geq 2$ cliques, and suppose that the algorithm generates a clique tree for any chordal graph with fewer than m cliques. Let K and P be, respectively, the first leaf clique and the first "parent" clique chosen by the algorithm, so that $\{K, P\}$ is the edge added to $\boldsymbol{\mathcal{E}}$ during the first step. From the way the edges are chosen, K is necessarily a leaf of T. Let $T' = T \setminus \{K\}$. Clearly, T' is the tree generated by subsequent steps of the algorithm. Moreover, steps 2–m of the algorithm are precisely the same as applying the algorithm directly to the graph $G \setminus Sim(K)$ with no prior elimination step. It follows from the induction hypothesis that T' is a clique tree of the chordal graph $G \setminus Sim(K)$. Consequently, Corollary 1 holds in T for every pair of cliques taken from $\boldsymbol{\mathcal{K}}_G - \{K\}$. All that remains to be shown is that for every clique $K' \in \boldsymbol{\mathcal{K}}_G - \{K\}$, the intersection $K \cap K'$ is contained in every clique on the path connecting K and K' in T. The statement is vacuously true if $K' = P$; thus we assume $K' \neq P$. Since $K \in \boldsymbol{\mathcal{L}}_G$ and $Sep_G(K) \subset P$ (see line 5 of the algorithm), we have

$$K \cap K' \subseteq Sep_G(K) \cap K' \subseteq P \cap K'.$$

Moreover, by the induction hypothesis, $P \cap K'$ is contained in every clique on the path connecting P with K', which obtains the result. ■

4.2. Incorporating maximum cardinality leaf sets in the algorithm

The new minimum-diameter clique tree algorithm is a special case of the algorithm in Figure 4.1. It is however a bit more complicated, with an outer loop that selects a maximum cardinality leaf set $\boldsymbol{\mathcal{L}}_{max}$ to be removed at each major step, and an inner loop that removes the members of $\boldsymbol{\mathcal{L}}_{max}$ one after another in arbitrary order. Clearly, this approach is based on the assumption that elimination of a clique $K \in \mathcal{L}_{max}$ by an iteration of the inner loop causes *none* of the uneliminated cliques $K' \in \mathcal{L}_{max}$ to become a non-leaf in the reduced graph. To address this issue, let $T \in \mathcal{T}_G$ and $K \in \mathcal{L}_T$, and consider the reduced graph $G' = G \backslash Sim_G(K)$ and the tree $T' = T \backslash \{K\}$. The following simple lemma contains the properties of G' and T' needed to deal with this issue and other closely related issues associated with our algorithm. (The fourth property is needed elsewhere, but it is most convenient to include it here.)

Lemma 9. *Let $T \in \mathcal{T}_G$ and $K \in \mathcal{L}_T$, and consider $T' = T \setminus \{K\}$ and $G' = G \setminus Sim_G(K)$. The following properties hold for G, T, G' and T':*

1. $T' \in \mathcal{T}_{G'}$.
2. $\mathcal{L}_T - \{K\} \subseteq \mathcal{L}_{T'} \subseteq \mathcal{L}_{G'}$.
3. *For* $K' \in \mathcal{L}_T - \{K\}$, $Sim_{G'}(K') = Sim_G(K')$ *and* $Sep_{G'}(K') = Sep_G(K')$.
4. $\mathcal{M}_{G'} = \mathcal{M}_G - \{Sep_G(K)\}$.

Proof. Let $K', K'' \in \mathcal{K}_{G'} = \mathcal{K}_G - \{K\}$. From the definition of T and T', it follows that the path connecting K' and K'' in T' is identical to the one connecting the pair in T. Thus, from Corollary 1 (applied to $T \in \mathcal{T}_G$) we have $T' \in \mathcal{T}_{G'}$. Clearly, any leaf in $\mathcal{L}_T - \{K\}$ remains a leaf in T', and thus $\mathcal{L}_T - \{K\} \subseteq \mathcal{L}_{T'} \subseteq \mathcal{L}_{G'}$.

Choose $K' \in \mathcal{L}_T - \{K\}$ and let $\{K', K''\}$ be the single edge incident on K' in T. Since $\{K', K''\}$ is also the single edge incident on K' in T', it follows from Lemma 1 that $K' \cap K'' = Sep_{G'}(K') = Sep_G(K')$. Futhermore, since $|K'|$ is unchanged and $K' \in \mathcal{L}_{T'}$, by Lemma 7 we have $Sim_{G'}(K') = Sim_G(K')$. Finally, the fourth property (i.e., $\mathcal{M}_{G'} = \mathcal{M}_G - \{Sep_G(K)\}$) is an immediate consequence of property 1 and the definition of separator multisets. ■

Our algorithm for computing a minimum-diameter clique tree is shown in Figure 4.2. At the beginning of each major step (i.e., each iteration of the **while** loop), H is the chordal graph remaining to be eliminated. After selecting a maximum cardinality leaf set $\mathcal{L}_{max}$ in line 4, the inner loop (lines 5–9) eliminates the leaf cliques in $\mathcal{L}_{max}$ one at a time in some arbitrary order.

Let $T \in \mathcal{T}_G$ be a clique tree for which $\mathcal{L}_T$ is the leaf set $\mathcal{L}_{max} \subseteq \mathcal{L}_G$ chosen for elimination during the *first* iteration of the **while** loop. Applying Lemma 9 to T, we justify several details found in the inner loop with the following remarks.

Remark 1. Line 6 assumes the existence of an appropriate "parent" $P \in \mathcal{K}_H - \mathcal{L}_{max}$ for each leaf $K \in \mathcal{L}_{max}$. For each $K \in \mathcal{L}_{max}$, the single clique P adjacent to $K \in \mathcal{L}_T$ is such a parent.

```
01 𝓔_min ← ∅
02 H ← G; H' ← G
03 while |𝓚_H| ≥ 3 do
04       Choose a maximum cardinality leaf set 𝓛_max ⊆ 𝓛_H
05       for K ∈ 𝓛_max do
06             Choose P ∈ 𝓚_H − 𝓛_max that also contains Sep_H(K)
07             𝓔_min ← 𝓔_min ∪ {K, P}
08             H' ← H' \ Sim_H(K)
09       end for
10       H ← H'
11 end while
12 if |𝓚_H| = 2 do
13       𝓔_min ← 𝓔_min ∪ {K, P} where {K, P} = 𝓚_H.
14 end if
```

Figure 4.2: Algorithm for generating a minimum-diameter clique tree.

Remark 2. Properties 1 and 2 ensure that after the removal of a leaf clique in the inner loop, the uneliminated members of $\boldsymbol{\mathcal{L}}_{max}$ remain leaves in the reduced graph, as required during subsequent iterations.

Remark 3. Properties 1 and 3 ensure that $Sep_{H'}(K) = Sep_H(K)$ and $Sim_{H'}(K) = Sim_H(K)$ for each leaf clique $K \in \boldsymbol{\mathcal{L}}_{max}$ in the reduced chordal graph. In other words, not only do the leaves in $\boldsymbol{\mathcal{L}}_{max}$ remain leaves as the inner loop progresses through the elimination steps, they also retain the same separator and simplicial node sets that they had when chosen for elimination at the beginning of the major step. The invariance of these two sets is used explicitly in lines 6 and 8 of the algorithm.

The following lemma plays a key role in the next subsection where we prove that any clique tree generated by the algorithm has minimum diameter.

Lemma 10. *The algorithm in Figure 4.2 generates a clique tree T for which $\boldsymbol{\mathcal{L}}_T = \boldsymbol{\mathcal{L}}_{max} \subseteq \boldsymbol{\mathcal{L}}_G$ is the maximum cardinality leaf set eliminated by the first major step of the algorithm.*

Proof. As in the proof of Lemma 8, we leave it for the reader to verify that the new algorithm generates a tree. It follows from Remark 2 that the new algorithm is a special case of the algorithm in Figure 4.1, and therefore by Lemma 8 the tree T_{alg} generated by the algorithm is a clique tree. Consider the maximum cardinality leaf set $\boldsymbol{\mathcal{L}}_{max} \subseteq \boldsymbol{\mathcal{L}}_G$ eliminated by the first major step of the algorithm. During the first major step, the algorithm adds to $\boldsymbol{\mathcal{E}}_{min}$ precisely one edge incident on K for each $K \in \boldsymbol{\mathcal{L}}_{max}$. Since each clique $K \in \boldsymbol{\mathcal{L}}_{max}$ is eliminated during this first step, none of the edges added during subsequent steps can be incident on K. It follows that each clique in $\boldsymbol{\mathcal{L}}_{max}$ is adjacent to only one clique in T_{alg}, and thus $\boldsymbol{\mathcal{L}}_{max} \subseteq \boldsymbol{\mathcal{L}}_{T_{alg}}$. Since $\boldsymbol{\mathcal{L}}_{max}$ is a maximum cardinality leaf set, $\boldsymbol{\mathcal{L}}_{T_{alg}} = \boldsymbol{\mathcal{L}}_{max}$, which concludes the proof. ∎

4.3. Diameter minimization in the new algorithm

For any clique tree $T \in \boldsymbol{\mathcal{T}}_G$ and any pair of cliques $K, K' \in \boldsymbol{\mathcal{K}}_G$, let $d_T(K, K')$ be the distance from K to K' along the single path connecting the pair in T. The diameter of T is given by $diam(T) := \max\{d_T(K, K')\}$, where K and K' range over every distinct pair of cliques taken from $\boldsymbol{\mathcal{K}}_G$. Of course, it suffices to limit the range of K and K' to every distinct pair of leaves taken from $\boldsymbol{\mathcal{L}}_T$. This section proves that the algorithm in Figure 4.2, in fact, finds a clique tree T_{min} that minimizes $diam(T)$ over all clique trees $T \in \boldsymbol{\mathcal{T}}_G$. This is, of course, equivalent to finding a clique tree of minimum height. To proceed, we show in Lemma 11 that when $P = P'$ in the proof of Lemma 2 (see Figure 3.1), the diameter of the new tree is no more than the diameter of the original tree. We then show that for any maximum cardinality leaf set $\boldsymbol{\mathcal{L}}_{max}$, there exists a minimum-diameter clique tree $T_{min} \in \boldsymbol{\mathcal{T}}_G$ for which $\boldsymbol{\mathcal{L}}_{T_{min}} = \boldsymbol{\mathcal{L}}_{max}$, after which the main result follows by a simple induction argument.

Lemma 11. *Assume $|\boldsymbol{\mathcal{K}}_G| \geq 3$. Let $K \in \boldsymbol{\mathcal{L}}(S)$ and suppose K is not a leaf of the clique tree $T \in \boldsymbol{\mathcal{T}}_G$. Let P be a neighbor of K in T such that $K \cap P = S$. Then there exists a clique tree $T' \in \boldsymbol{\mathcal{T}}_G$ for which the following properties hold:*

1. *K is a leaf of T'.*
2. *The sole difference between T and T' is that each edge $\{C, K\}$ incident on K in T, with the exception $\{P, K\}$, has been replaced with the edge $\{C, P\}$ in T'.*
3. *$diam(T') \leq diam(T)$.*

Proof. First, note that the existence of a neighbor P of K in T for which $K \cap P = S = Sep(K)$ is ensured by Lemma 1. Now consider the restructured clique tree T' produced in the proof of Lemma 2 when $P = P'$ (see Figure 4.3). That the first two properties hold for T and T' follows directly from the proof of Lemma 2. To verify the third property, first note that the only paths whose lengths are longer in T' than they are in T are those connecting K to some node K' in one of the moved subtrees. Moreover, the path connecting K and K' in the restructured tree is no longer than the path connecting K' and P in the original tree (see Figure 4.3). It follows that making all the neighbors of K (except P) neighbors of P cannot increase the diameter. ∎

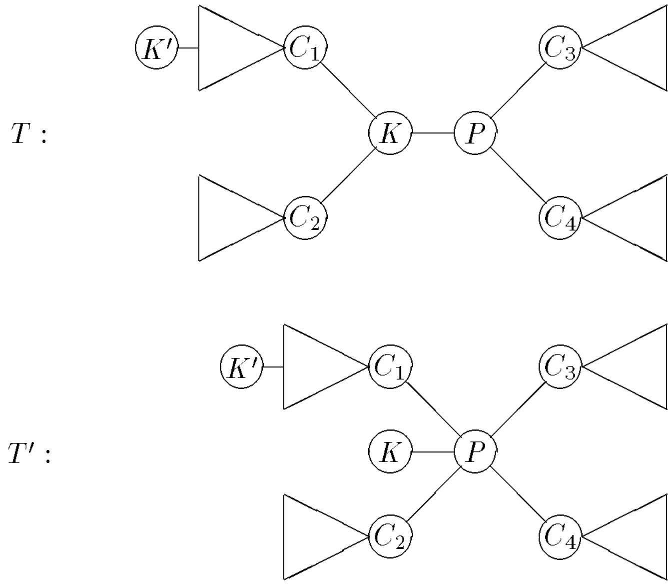

Figure 4.3: Transformation of T into T' in which K is a leaf and for which $diam(T') \leq diam(T)$, as discussed in the proof of Lemma 11.

Lemma 12. *Assume $|\boldsymbol{\mathcal{K}}_G| \geq 3$ and let $T \in \boldsymbol{\mathcal{T}}_G$ be any clique tree for which $|\boldsymbol{\mathcal{L}}_T| \geq |\boldsymbol{\mathcal{L}}_{T'}|$ for all $T' \in \boldsymbol{\mathcal{T}}_G$. Then there exists a minimum-diameter clique tree $T_{min} \in \boldsymbol{\mathcal{T}}_G$ such that $\boldsymbol{\mathcal{L}}_{T_{min}} = \boldsymbol{\mathcal{L}}_T$.*

Proof. Let $T \in \boldsymbol{\mathcal{T}}_G$ be chosen as in the premise. Choose a minimum-diameter clique tree $T_{min} \in \boldsymbol{\mathcal{T}}_G$ for which $\boldsymbol{\mathcal{L}}_{T_{min}}$ contains as many of the leaf cliques belonging to $\boldsymbol{\mathcal{L}}_T$ as possible. By way of contradiction, assume that $\boldsymbol{\mathcal{L}}_{T_{min}} \neq \boldsymbol{\mathcal{L}}_T$. Since $|\boldsymbol{\mathcal{L}}_T| \geq |\boldsymbol{\mathcal{L}}_{T_{min}}|$, there exists a leaf clique $K \in \boldsymbol{\mathcal{L}}_T - \boldsymbol{\mathcal{L}}_{T_{min}}$. Without loss of generality, suppose that $K \in \boldsymbol{\mathcal{L}}(S)$. Since $|\boldsymbol{\mathcal{K}}_G| \geq 3$ and $|\boldsymbol{\mathcal{K}}(S)| \geq 2$, at least one clique $K' \in \boldsymbol{\mathcal{K}}(S)$ is *not* a leaf in T.

Now consider the subtree of T_{min} induced by $\boldsymbol{\mathcal{K}}(S)$. Let $P \in \boldsymbol{\mathcal{K}}(S)$ be the clique adjacent to K along the path from K to K' in the subtree of T_{min} induced by $\boldsymbol{\mathcal{K}}(S)$ (possibly $P = K'$). Observe that if $P = K'$, then P is not a leaf of T, and if $P \neq K'$, then P is not a leaf of T_{min}. It follows that P is not one of the leaf cliques that T and T_{min} have in common. Thus, using Lemma 11 to restructure T_{min} (see Figure 4.3) results in a clique tree also of minimum diameter, but with one more leaf clique K in common with T than originally possessed by T_{min}. This is contrary to our assumption about T_{min}, which concludes the proof. ■

Recall that the tree obtained by pruning the set of leaves $\boldsymbol{\mathcal{L}}_T$ from $T \in \boldsymbol{\mathcal{T}}_H$ is denoted by $T \setminus \boldsymbol{\mathcal{L}}_T$. We let $Sim_H(\boldsymbol{\mathcal{L}}_T)$ be the union of all simplicial node sets $Sim_H(K)$ where $K \in \boldsymbol{\mathcal{L}}_T$. We are now ready to prove our main result.

Theorem 4.1. *The algorithm in Figure 4.2 generates a clique tree of minimum diameter.*

Proof. That the algorithm generates a clique tree was proven in Lemma 10. The proof that the clique tree has minimum diameter is by induction on $m = |\mathcal{K}_G|$. The base steps $m = 1$ and $m = 2$ are trivial. Let G be a chordal graph with $m \geq 3$ cliques and assume that the algorithm minimizes clique tree diameter for any chordal graph with fewer cliques. Let T_{alg} be a clique tree generated by the algorithm.

By Lemma 10, $\mathcal{L}_{T_{alg}}$ is the maximum cardinality leaf set $\mathcal{L}_{max} \subseteq \mathcal{L}_G$ chosen for elimination during the first major step of the algorithm. Remarks 2 and 3 in the previous subsection imply that the first major step eliminates the nodes in $Sim_G(\mathcal{L}_{T_{alg}})$. Clearly, $T_{alg} \setminus \mathcal{L}_{T_{alg}}$ is the tree generated by subsequent major steps of the algorithm. Moreover, these subsequent steps are precisely the same as applying the algorithm directly to the graph $G \setminus Sim_G(\mathcal{L}_{T_{alg}})$ with no prior elimination step. It follows from the induction hypothesis that $T_{alg} \setminus \mathcal{L}_{T_{alg}}$ is a *minimum-diameter* clique tree of the chordal graph $G \setminus Sim_G(\mathcal{L}_{T_{alg}})$.

By Lemma 12, there exists a minimum-diameter clique tree $T_{min} \in \mathcal{T}_G$ such that $\mathcal{L}_{T_{min}} = \mathcal{L}_{T_{alg}}$. Thus, $T_{alg} \setminus \mathcal{L}_{T_{alg}}$ and $T_{min} \setminus \mathcal{L}_{T_{min}}$ are both clique trees of $G \setminus Sim_G(\mathcal{L}_{T_{alg}})$. It follows that

$$diam(T_{alg} \setminus \mathcal{L}_{T_{alg}}) \leq diam(T_{min} \setminus \mathcal{L}_{T_{min}}).$$

Note that whenever $m \geq 3$, elimination of all leaves of any clique tree results in a tree whose diameter has been reduced by two. Thus we can write

$$\begin{aligned} diam(T_{alg}) &= diam(T_{alg} \setminus \mathcal{L}_{T_{alg}}) + 2 \\ &\leq diam(T_{min} \setminus \mathcal{L}_{T_{min}}) + 2 \\ &= diam(T_{min}), \end{aligned}$$

which proves the result. ∎

5. A linear time implementation

With careful attention to details, we can devise a linear time implementation of our minimum-diameter clique tree algorithm (i.e., an $O(|V| + |E|)$ implementation). The order in which the cliques are eliminated from H can be viewed as a block variant of the Jess and Kees ordering. We refer the reader to [20, 23, 25] for a description of 1) the basic Jess and Kees ordering scheme, 2) its use in minimizing the height of the *elimination tree*[5] associated with a sparse Cholesky factor [25], and 3) two implementations of the algorithm [23, 25]. Lewis et al. [23] describe a very efficient implementation of the algorithm in great detail. Many of the techniques used in their implementation are also helpful in devising an efficient implementation of the minimum-diameter clique tree algorithm in Figure 4.2. Where appropriate, we refer the reader to [23] for details that apply to both algorithms.[6]

This section discusses several implementation issues specific to the minimum-diameter clique tree algorithm. Section 5.1 presents an efficient algorithm for selecting a maximum cardinality leaf set $\boldsymbol{\mathcal{L}}_{max} \subseteq \boldsymbol{\mathcal{L}}_H$ in line 4 of Figure 4.2. Section 5.2 introduces a detailed version of the minimum-diameter clique tree algorithm and verifies that it correctly implements the algorithm in Figure 4.2. Finally, Section 5.3 demonstrates the linear time complexity of the new algorithm.

5.1. Selection of a maximum cardinality leaf set

Consider the problem of selecting a maximum cardinality leaf set $\boldsymbol{\mathcal{L}}_{max} \subseteq \boldsymbol{\mathcal{L}}_H$ in line 4 of Figure 4.2. (Computing $\boldsymbol{\mathcal{L}}_H$ will be discussed later.) The algorithm to calculate $\boldsymbol{\mathcal{L}}_{max}$, shown in Figure 5.1, considers each clique in $\boldsymbol{\mathcal{L}}_H$ in some arbitrary order. To

$H' \leftarrow H$
$\boldsymbol{\mathcal{L}}_{max} \leftarrow \emptyset$
for $K \in \boldsymbol{\mathcal{L}}_H$ **do**
    if $\sigma_{H'}(K) = \sigma_H(K)$ **do**
        $\boldsymbol{\mathcal{L}}_{max} \leftarrow \boldsymbol{\mathcal{L}}_{max} \cup \{K\}$
        $H' \leftarrow H' \setminus Sim_H(K)$
    end if
end for

Figure 5.1: Algorithm for generating a maximum cardinality leaf set.

test $K \in \boldsymbol{\mathcal{L}}_H$ for inclusion in $\boldsymbol{\mathcal{L}}_{max}$, the algorithm checks to see if there has been no change in the parameter $\sigma(K)$. (That is, does $\sigma_{H'}(K) = \sigma_H(K)$?) The remainder of this subsection is devoted to proving that this test can be used to obtain a maximum cardinality leaf set $\boldsymbol{\mathcal{L}}_{max} \subseteq \boldsymbol{\mathcal{L}}_H$. First, Lemma 13 gives a useful condition that holds if

[5] Note that the elimination tree is *not* a clique tree, and minimizing its height moreover does not minimize the height of the associated clique tree.

[6] The notation used in [23] differs a great deal from that found in this paper. For example, their *ancestor* sets, $Anc(K)$, are the separators, and $S(K)$ contains the simplicial nodes of K. Also note that their definition of clique trees is more restrictive than the one we are using.

and only if $\sigma_{H'}(K) = \sigma_H(K)$, and then Theorem 5.1 proves the algorithm in Figure 5.1 correct.

Lemma 13. *Let $S \in \overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_H)$ and choose $K \in \boldsymbol{\mathcal{L}}_H(S) \subseteq \boldsymbol{\mathcal{L}}_H$. When K is processed by line 4 of the algorithm in Figure 5.1, $\sigma_{H'}(K) = \sigma_H(K)$ if and only if $|\boldsymbol{\mathcal{K}}_{H'}(S)| \geq 2$.*

Proof. Let $K \in \boldsymbol{\mathcal{L}}_H(S) \subseteq \boldsymbol{\mathcal{L}}_H$, and consider the iteration of the algorithm that processes K. By definition, $\sigma_H(K) = |S|$. If $|\boldsymbol{\mathcal{K}}_{H'}(S)| \geq 2$, then by Lemma 6, $\sigma_{H'}(K) \geq |S|$. Since $\boldsymbol{\mathcal{K}}_{H'} \subset \boldsymbol{\mathcal{K}}_H$, it follows from Lemma 6 that $\sigma_{H'}(K) \leq \sigma_H(K) = |S|$. Thus, $\sigma_{H'}(K) = \sigma_H(K)$.

By Lemma 6 if $\sigma_{H'}(K) = \sigma_H(K)$, then $|K \cap K'| = |S|$ for some clique $K' \in \boldsymbol{\mathcal{K}}_{H'} - \{K\}$. From Corollary 1 and the fact that $\overline{\boldsymbol{\mathcal{S}}}_H(K) = \{S\}$, we have $K \cap K'' \subseteq S$ for every clique $K'' \in \boldsymbol{\mathcal{K}}_H - \{K\}$. Consequently, $K \cap K' = S$, and therefore $K, K' \in \boldsymbol{\mathcal{K}}_{H'}(S)$, which proves the result. ∎

Theorem 5.1. *The algorithm in Figure 5.1 computes a maximum cardinality leaf set $\boldsymbol{\mathcal{L}}_{max}$.*

Proof. Consider the partition of $\boldsymbol{\mathcal{L}}_H$ into leaf cohort sets,

$$\boldsymbol{\mathcal{L}}_H = \{\boldsymbol{\mathcal{L}}_H(S) \mid S \in \overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_H)\}.$$

Lemma 5 gives the two conditions that must be satisfied by $\boldsymbol{\mathcal{L}}_{max}$: 1) when $\boldsymbol{\mathcal{L}}_H(S) \subset \boldsymbol{\mathcal{K}}_H(S)$ every clique in $\boldsymbol{\mathcal{L}}_H(S)$ must be included in $\boldsymbol{\mathcal{L}}_{max}$, and 2) when $\boldsymbol{\mathcal{L}}_H(S) = \boldsymbol{\mathcal{K}}_H(S)$ precisely one clique in $\boldsymbol{\mathcal{L}}_H(S)$ must be excluded from $\boldsymbol{\mathcal{L}}_{max}$. We will consider an arbitrary leaf cohort set $\boldsymbol{\mathcal{L}}_H(S) = \{K_1, K_2, \ldots, K_t\} \subseteq \boldsymbol{\mathcal{L}}_H$, with the cliques listed in the order in which the algorithm processes them. (That cliques from other leaf cohort sets may be processed between two neighboring cliques in the list will have no bearing on the argument.)

First, note that $|\boldsymbol{\mathcal{K}}_{H'}(S)| \geq 2$ when the algorithm processes K_i, $1 \leq i \leq t-1$. Therefore, by Lemma 13, $\sigma_{H'}(K_i) = \sigma_H(K_i)$, and K_i is included in $\boldsymbol{\mathcal{L}}_{max}$. Now consider whether or not the algorithm includes K_t in $\boldsymbol{\mathcal{L}}_{max}$. There are two cases to consider. First, suppose $\boldsymbol{\mathcal{L}}_H(S) = \boldsymbol{\mathcal{K}}_H(S)$. It follows that $\boldsymbol{\mathcal{K}}_{H'}(S) = \{K_t\}$ when the algorithm finally examines K_t. Consequently by Lemma 13, $\sigma_{H'}(K_t) \neq \sigma_H(K_t)$, and therefore K_t is *not* selected, as desired.

Now, suppose $\boldsymbol{\mathcal{L}}_H(S) \subset \boldsymbol{\mathcal{K}}_H(S)$ and consider the following two subcases. First, assume $\boldsymbol{\mathcal{K}}_H(S) \nsubseteq \boldsymbol{\mathcal{L}}_H$. In this case, $|\boldsymbol{\mathcal{K}}_{H'}(S)| \geq 2$ when the algorithm examines K_t, and by Lemma 13, K_t is selected as desired. Now, assume $\boldsymbol{\mathcal{K}}_H(S) \subseteq \boldsymbol{\mathcal{L}}_H$. Let $K' \in \boldsymbol{\mathcal{L}}_H$ be chosen so that $K' \in \boldsymbol{\mathcal{K}}_H(S) - \boldsymbol{\mathcal{L}}_H(S)$. It follows that $K' \in \boldsymbol{\mathcal{L}}_H(S')$ where $S \subset S'$. The key to the proof is to choose K' that maximizes $|S'|$ among all the leaf separators $S' \in \overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_H)$ for which $S \subset S'$. Since $S \subset S'$, we have $\boldsymbol{\mathcal{K}}_H(S') \subseteq \boldsymbol{\mathcal{K}}_H(S) \subseteq \boldsymbol{\mathcal{L}}_H$. It suffices to show that $\boldsymbol{\mathcal{K}}_H(S') = \boldsymbol{\mathcal{L}}_H(S')$, for we have shown in the previous paragraph that if

$\mathcal{K}_H(S') = \mathcal{L}_H(S')$, then the "last" member of $\mathcal{L}_H(S')$ to be processed by the algorithm is excluded from $\mathcal{L}_{max}$ and thus retained in the graph H' when K_t is examined by the algorithm. Consequently, $|\mathcal{K}_{H'}(S)| \geq 2$, and therefore K_t is included in $\mathcal{L}_{max}$ as required. To verify that $\mathcal{L}_H(S') = \mathcal{K}_H(S')$, assume that $K'' \in \mathcal{K}_H(S') - \mathcal{L}_H(S')$. It follows that $K'' \in \mathcal{L}_H(S'')$ where $S' \subset S''$, contrary to the maximality of $|S'|$. Thus, we have $\mathcal{K}_H(S') = \mathcal{L}_H(S')$, which concludes the proof. ■

5.2. Implementation details

We now turn our attention to a detailed version of the minimum-diameter clique tree algorithm. This algorithm, presented in Figure 5.2, is essentially an expanded version of the algorithm in Figure 4.2. Practically missing altogether from the short version is the initialization phase comprising lines 1–8 in Figure 5.2. Line 4 in Figure 4.2 has been expanded into lines 10–21, which implement the algorithm for finding $\mathcal{L}_{max}$ displayed in Figure 5.1 and also build a data structure used to record the cliques in $\mathcal{L}_{max}$ and the new edges of the minimum-diameter clique tree. Using this data structure, lines 22–30 implement lines 6 and 7 of Figure 4.2, as well as construct the leaf set $\mathcal{L}_H$ for the next iteration through the main loop.

The remainder of this subsection takes a closer look at the detailed version of our minimum-diameter clique tree algorithm and gives the arguments needed to prove that it does indeed correctly implement the algorithm in Figure 4.2. We begin with a description of the method used to represent the "current" chordal graph. After that, we focus on the correctness of the **while** loop (lines 3–10 in Figure 4.2, lines 9–32 in Figure 5.2) that constitutes the bulk of the algorithm. In Section 5.2.2 we address the question of correctness for one iteration through the **while** loop, assuming all data are correct as the iteration begins. Then in Section 5.2.3 we argue that the data are correct at the beginning of each iteration through the **while** loop.

5.2.1. Representation of the chordal graph

The algorithm maintains a clique tree $T' \in \mathcal{T}_{H'}$ and *no other representation* of the reduced chordal graphs (lines 1, 13, 18); the chordal graphs H and H' are "updated" in the comments strictly for notational purposes (lines 1, 18, 31).

Initially, $T' \in \mathcal{T}_G$. (Details on an appropriate choice of $T' \in \mathcal{T}_G$ are provided in [23].) The clique tree T' is updated as each leaf clique is eliminated, much as H' is updated in Figure 5.1 as the leaf cliques are eliminated. An elimination step performed during the **for** loop in lines 11–21 removes $K \in \mathcal{L}_H$ from T' to obtain the next clique tree $T' \in \mathcal{T}_{H'}$ where H' is the new reduced graph $H' \leftarrow H' \backslash Sim_{H'}(K)$ (line 18). When $K \in \mathcal{L}_{T'}$, the reduced clique tree is computed by simply "pruning" K from T' (i.e., $T' \leftarrow T' \setminus \{K\}$). However, when $K \notin \mathcal{L}_{T'}$, the operation required to compute the new clique tree is a bit more complicated and requires some restructuring of the tree. We will let $T' \setminus_r \{K\}$

[**Input:** $T \in \boldsymbol{\mathcal{T}}_G$ and $\sigma_G(K), \rho_G(K)$ for $K \in \boldsymbol{\mathcal{K}}_G$]
[**Output:** A minimum-diameter clique tree $T_{min} = (\boldsymbol{\mathcal{K}}_G, \boldsymbol{\mathcal{E}}_{min}) \in \boldsymbol{\mathcal{T}}_G$]
$T' \leftarrow T$ [$H \leftarrow G$; $H' \leftarrow G$]
$\boldsymbol{\mathcal{E}}_{min} \leftarrow \emptyset$; $\boldsymbol{\mathcal{L}}_H \leftarrow \emptyset$
for $K \in \boldsymbol{\mathcal{K}}_H$ **do**   [initialization]
    $\sigma_{H'}(K) \leftarrow \sigma_G(K)$; $\sigma_H(K) \leftarrow \sigma_G(K)$
    $\rho_{H'}(K) \leftarrow \rho_G(K)$; $\rho_H(K) \leftarrow \rho_G(K)$
    if $\sigma_H(K) + \rho_H(K) = |K|$ **then** $\boldsymbol{\mathcal{L}}_H \leftarrow \boldsymbol{\mathcal{L}}_H \cup \{K\}$
    $\boldsymbol{\mathcal{C}}(K) \leftarrow \emptyset$
end for
while $|\boldsymbol{\mathcal{K}}_H| \geq 3$ **do**   [main loop]
    $\boldsymbol{\mathcal{P}} \leftarrow \emptyset$
    for $K \in \boldsymbol{\mathcal{L}}_H$ **do**   [build $\boldsymbol{\mathcal{L}}_{max}$ in sets $\boldsymbol{\mathcal{C}}(P)$, $P \in \boldsymbol{\mathcal{P}}$]
        if $\sigma_{H'}(K) = \sigma_H(K)$ **do**   [include K in $\boldsymbol{\mathcal{L}}_{max}$]
            Select $\{K, P\} \in \boldsymbol{\mathcal{E}}_{T'}$ for which $|K \cap P| = \sigma_H(K)$
            $\boldsymbol{\mathcal{P}} \leftarrow \boldsymbol{\mathcal{P}} \cup \{P\}$ [include P in set of parents]
            if $K \in \boldsymbol{\mathcal{P}}$ **then** $\boldsymbol{\mathcal{P}} \leftarrow \boldsymbol{\mathcal{P}} - \{K\}$
            $\boldsymbol{\mathcal{C}}(P) \leftarrow \boldsymbol{\mathcal{C}}(P) \cup \{K\} \cup \boldsymbol{\mathcal{C}}(K)$ [update children of P]
            $\boldsymbol{\mathcal{C}}(K) \leftarrow \emptyset$
            $T' \leftarrow T' \setminus_r \{K\}$ [$H' \leftarrow H' \setminus Sim_H(K)$]
            Update $\sigma_{H'}(P)$ and $\rho_{H'}(P)$
        end if
    end for
    $\boldsymbol{\mathcal{L}}_H \leftarrow \emptyset$
    for $P \in \boldsymbol{\mathcal{P}}$ **do**   [prepare for next iteration of the **while** loop]
        for $K \in \boldsymbol{\mathcal{C}}(P)$ **do**   [record new edges of T_{min}]
            $\boldsymbol{\mathcal{E}}_{min} \leftarrow \boldsymbol{\mathcal{E}}_{min} \cup \{K, P\}$
        end for
        $\boldsymbol{\mathcal{C}}(P) \leftarrow \emptyset$
        $\sigma_H(P) \leftarrow \sigma_{H'}(P)$; $\rho_H(P) \leftarrow \rho_{H'}(P)$
        if $\sigma_H(P) + \rho_H(P) = |P|$ **then** $\boldsymbol{\mathcal{L}}_H \leftarrow \boldsymbol{\mathcal{L}}_H \cup \{P\}$
    end for
    [$H \leftarrow H'$]
end while
if $|\boldsymbol{\mathcal{K}}_H| = 2$ **do**
    $\boldsymbol{\mathcal{E}}_{min} \leftarrow \boldsymbol{\mathcal{E}}_{min} \cup \{K, K'\}$ where $\{K, K'\} = \boldsymbol{\mathcal{K}}_H$.
end if

Figure 5.2: Detailed algorithm for generating a minimum-diameter clique tree.

denote the tree obtained by first performing the appropriate restructuring of T' and then removing the leaf clique K from the tree. The required restructuring operation (needed to transform K into a leaf) has already been introduced in Lemma 11 and illustrated in Figure 4.3. Based on this restructuring operation, $T' \setminus_r \{K\}$ is illustrated in Figure 5.3.

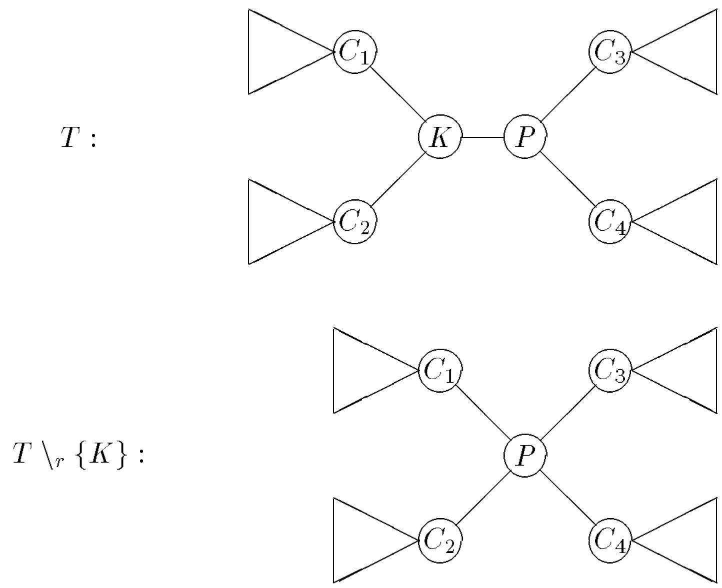

Figure 5.3: Transformation of T into $T \setminus_r \{K\}$.

Lewis et al. [23] used this technique for eliminating an arbitrary clique $K \in \boldsymbol{\mathcal{L}}_H$ from a clique tree $T' \in \boldsymbol{\mathcal{T}}_{H'}$, in which K may or may not be a leaf. The interested reader should consult [23] for a detailed description of the data structure required to implement it efficiently. Section 5.3 contains a brief discussion of the more important features of this data structure and how it is used.

5.2.2. Correct implementation of one iteration of the while loop

Lines 10–21 and 24–26 in Figure 5.2 are used to actually implement the operations within the **while** loop in Figure 4.2. (All other statements inside the **while** loop in Figure 5.2 are used to update data for the *next* iteration through the **while** loop.) As stated earlier, the **for** loop in lines 11–21 implements the algorithm for finding $\boldsymbol{\mathcal{L}}_{max}$ displayed in Figure 5.1 and also builds a data structure used to collect the cliques in $\boldsymbol{\mathcal{L}}_{max}$ and the new clique tree edges to be placed in $\boldsymbol{\mathcal{E}}_{min}$ by lines 24–26.

To see that the loop in lines 11–21 correctly implements the algorithm in Figure 5.1, observe that lines 3, 4, 6, 7, and 8 of Figure 5.1 correspond directly to lines 11, 12, 18, 20, and 21 respectively of Figure 5.2. Lines 13–17 of Figure 5.2 manipulate the data structure for collecting the leaves and new edges, as well as perform the operation in line 5 of the algorithm in Figure 5.1.

Two issues connected with operations performed by this loop need further discussion. First, note that after the algorithm eliminates a leaf clique K in line 18, the only parameters updated are $\sigma_{H'}(P)$ and $\rho_{H'}(P)$ in line 19. We must show that line 19 is sufficient to maintain correct parameters $\sigma_{H'}(K')$ and $\rho_{H'}(K')$ for all cliques $K' \in \boldsymbol{\mathcal{K}}_{H'}$. Second, we must verify that the data structure correctly stores a maximum cardinality leaf set $\boldsymbol{\mathcal{L}}_{max}$ and a set of new edges connecting each member of $\boldsymbol{\mathcal{L}}_{max}$ to an appropriate "parent" clique.

The following proposition shows that with each graph reduction (line 18) the only clique K' for which the values of $\sigma_{H'}(K')$ or $\rho_{H'}(K')$ may have changed during the iteration of the **for** loop is the clique P. As a result, line 19 of Figure 5.2 correctly updates the only values $\sigma_{H'}(K')$ and $\rho_{H'}(K')$ ($K' \in \boldsymbol{\mathcal{K}}_{H'}$) that may require changing due to the elimination of K.

Proposition 1. *Let $K \in \boldsymbol{\mathcal{L}}_H(S)$ and $H' = H \setminus Sim(K)$. If $\sigma_{H'}(P) \neq \sigma_H(P)$ or $\rho_{H'}(P) \neq \rho_H(P)$ for any clique $P \in \boldsymbol{\mathcal{K}}_{H'}$, then $\boldsymbol{\mathcal{K}}_H(S) = \{K, P\}$.*

Proof. Assume $\sigma_{H'}(P) \neq \sigma_H(P)$. It follows that any separator $S' \in \overline{\boldsymbol{\mathcal{S}}}_H(P)$ for which $\sigma_H(P) = |S'|$ is not a member of $\boldsymbol{\mathcal{M}}_{H'}$. From property 4 of Lemma 9, the multiset of separators of the reduced graph $H' = H \setminus Sim(K)$ is given by $\boldsymbol{\mathcal{M}}_H - \{S\}$. It follows that S' is unique and moreover $S' = S$. By way of contradiction, assume $|\boldsymbol{\mathcal{K}}_{H'}(S)| \geq 2$. Then by Lemma 6, $\sigma_{H'}(P) \geq |S| = \sigma_H(P)$. However, since $\boldsymbol{\mathcal{K}}_{H'} = \boldsymbol{\mathcal{K}}_H - \{K\}$, by Lemma 6, $\sigma_H(P) \leq |S|$. Consequently, $\sigma_{H'}(P) = \sigma_H(P)$, contrary to our assumption that $\sigma_{H'}(P) \neq \sigma_H(P)$. Therefore $\boldsymbol{\mathcal{K}}_{H'}(S) = \{P\}$, which implies that $\boldsymbol{\mathcal{K}}_H(S) = \{K, P\}$.

Now assume $\rho_{H'}(P) \neq \rho_H(P)$. Since $Sim_H(P) \subseteq Sim_{H'}(P)$, the assumption implies that there are some new simplicial nodes in P, i.e., $Sim_H(P) \subset Sim_{H'}(P)$. Note that the nodes of H' that belong to fewer cliques than they did in H are precisely those in S, and they belong to precisely one less clique (due to the removal of K). As a result, the new simplicial nodes of P must come from S. If $|\boldsymbol{\mathcal{K}}_H(S)| \geq 3$, then removal of K from H would result in no new simplicial nodes at all. Consequently, $|\boldsymbol{\mathcal{K}}_H(S)| = 2$. Since new simplicial nodes appear in P, it follows that P must be the other clique of H that contains S, and thus $\boldsymbol{\mathcal{K}}_H(S) = \{K, P\}$. ∎

We now turn our attention to the data structure used to maintain $\boldsymbol{\mathcal{L}}_{max}$ and the new clique tree edges. This data structure is composed of the following sets:

1. Upon completion of the **for** loop in lines 11–21, the set $\boldsymbol{\mathcal{P}}$ contains all the cliques in the remaining chordal graph H' that will serve as *parents* of the leaf cliques eliminated by this major step.

2. Also upon completion of the loop, the set $\boldsymbol{\mathcal{C}}(P)$ ($P \in \boldsymbol{\mathcal{P}}$) contains all members of $\boldsymbol{\mathcal{L}}_{max}$ that will become *children* of P in T_{min}. That is, $\boldsymbol{\mathcal{L}}_{max}$ comprises the sets

$\mathcal{C}(P)$ $(P \in \mathcal{P})$, and for each $K \in \mathcal{C}(P)$, the edge $\{K, P\}$ will be added to $\mathcal{E}_{min}$.

These sets are computed as follows. Upon entry into the **for** loop, $\mathcal{P} = \emptyset$ (line 10) and $\mathcal{C}(K) = \emptyset$ for every clique $K \in \mathcal{K}_H$ (lines 7, 17 and 27). As the loop processes a clique $K \in \mathcal{L}_H$ that will be eliminated, it chooses in line 13 a neighbor P of K in the current clique tree T' that also contains $Sep_H(K)$. Lines 14–15 update $\mathcal{P}$ by adding P and removing K, if necessary. In line 16, K and the members of $\mathcal{C}(K)$ are merged into $\mathcal{C}(P)$, or loosely speaking , K and the "children" of K are made children of P. (This corresponds closely to the restructuring operation described in Lemma 11 and illustrated in Figure 4.3).

To show that the scheme works, we need to show that three properties hold *upon completion of the loop.* The first two are trivial; we leave it to the reader to confirm that

1. $\mathcal{P} \subseteq \mathcal{K}_{H'}$, and

2. The union of the disjoint sets $\mathcal{C}(P)$ $(P \in \mathcal{P})$ is a maximum cardinality leaf set $\mathcal{L}_{max}$ of H.

The following proposition states and proves the third required property.

Proposition 2. *Upon completion of the* **for** *loop in lines 11–21 in Figure 5.2, we have $Sep_H(K) \subset P$ for every clique $K \in \mathcal{C}(P)$ $(P \in \mathcal{P})$.*

Proof. The following simple induction argument suffices. The result holds vacuously before the first iteration of the loop is begun. Now we assume that it holds as an iteration of the loop begins, and will show that it holds when the iteration is completed. Let H' be the graph remaining as the iteration begins, K the member of $\mathcal{L}_H$ chosen for elimination, and P the selected neighbor of K in the current clique tree (line 13). Upon completion of the iteration, there is a new version of $\mathcal{C}(P)$ containing K, $\mathcal{C}(K)$, and those cliques belonging to $\mathcal{C}(P)$ at the beginning of the iteration. By the induction hypothesis, the property continues to hold for those cliques that were contained in $\mathcal{C}(P)$ at the beginning of the iteration. Line 13 of the algorithm implies that the property holds for K. Now, choose $C \in \mathcal{C}(K)$. Note that by property 2 above, $C \in \mathcal{L}_H$. By induction we have $Sep_H(C) \subseteq K$. Moreover, since $K \in \mathcal{L}_H$, we have $Sep_H(C) \subseteq Sep_H(K)$, which by line 13 of the algorithm is a subset of P. Consequently, $Sep_H(C) \subset P$, and thus the result holds for the new version of $\mathcal{C}(P)$. Finally, by induction the property continues to hold for the sets $\mathcal{C}(P')$, $P' \in \mathcal{P} - \{P\}$, none of which are modified during the iteration. ■

It follows from Proposition 2 and the discussion preceding it that the edges stored implicitly in the sets $\mathcal{P}$ and $\mathcal{C}(P)$ are precisely the edges that should be added to $\mathcal{E}_{min}$ in lines 6 and 7 of Figure 4.2. The detailed algorithm adds these edges to $\mathcal{E}_{min}$ in lines 24–26.

5.2.3. Correct data to begin each iteration of the while loop

We have described the most important features of the detailed version of our minimum-diameter clique tree algorithm, and have shown that lines 11–21 and 24–26 correctly implement the lines inside the main loop in Figure 4.2. All that remains to be shown is that the data structures and parameters are correct as each iteration of the main loop in Figure 5.2 begins.

We have already discussed how a clique tree representation of H' is maintained. More details on this issue can be found in the next subsection and in [23]. In addition, from Proposition 1 we know that the only cliques for which $\sigma_{H'}(K) \neq \sigma_H(K)$ or $\rho_{H'}(K) \neq \rho_H(K)$ at the beginning of the loop in lines 23–30 are the cliques belonging to $\boldsymbol{\mathcal{P}}$. Since line 28 updates $\sigma_H(K)$ and $\rho_H(K)$ for each clique $K \in \boldsymbol{\mathcal{P}}$, these values will be correct at the beginning of each iteration of the main loop. Thus, we need to show only that $\boldsymbol{\mathcal{L}}_H$ is correct at the beginning of each iteration.

Note that line 6 uses Lemma 7 to construct the initial list $\boldsymbol{\mathcal{L}}_H$. The next proposition justifies the use of lines 22 and 29 to construct the leaf set $\boldsymbol{\mathcal{L}}_H$ to be used in the next iteration.

Proposition 3. *Upon entering the* **for** *loop in lines 23–30 of Figure 5.2,* $\boldsymbol{\mathcal{L}}_{H'} \subset \boldsymbol{\mathcal{P}}$.

Proof. Assume the algorithm is entering the loop, and let $K \in \boldsymbol{\mathcal{L}}_{H'}$. If $K \in \boldsymbol{\mathcal{L}}_{H'} - \boldsymbol{\mathcal{L}}_H$, then it follows from Lemma 7 that $\sigma_{H'}(K) \neq \sigma_H(K)$ or $\rho_{H'}(K) \neq \rho_H(K)$, and by Proposition 1 we have $K \in \boldsymbol{\mathcal{P}}$. Now assume $K \in \boldsymbol{\mathcal{L}}_{H'} \cap \boldsymbol{\mathcal{L}}_H$. It follows that the algorithm excluded K from $\boldsymbol{\mathcal{L}}_{max}$ because $\sigma_{H'}(K) \neq \sigma_H(K)$ when K was considered for inclusion. So again by Proposition 1 we have $K \in \boldsymbol{\mathcal{P}}$. ∎

5.3. Complexity

To facilitate our discussion of the algorithm's time complexity, define $n := |V|$, $e := |E|$, $m := |\boldsymbol{\mathcal{K}}_G|$, and $q := \sum_{i=1}^{m} |K_i|$. (Recall that $G = (V, E)$.) It is well-known that $m \leq n$ and $q \leq e$ [12], and moreover in some practical applications $q \ll e$, as pointed out in [23].

An appropriate initial clique tree $T \in \boldsymbol{\mathcal{T}}_G$ can be computed in $O(n + e)$ time by applying a slightly modified version of the maximum cardinality search algorithm to the underlying chordal graph G [26, 30]. We should note however that the input clique tree is not obtained in this fashion in the sparse matrix application area. A clique tree stored in an appropriate data structure can be obtained very efficiently more or less as a by-product of a data structure generated in the course of solving the linear system at hand (see [23]).

A singly-linked list suffices to represent $\boldsymbol{\mathcal{L}}_H$ throughout the computation (see lines 2, 6, 11, 22, 29). Thus, the time complexity of the initialization loop (lines 3–8) is $O(m)$.

Consider the **for** loop in lines 11–21 *excluding lines 13, 18, and 19.* (These three lines are discussed later.) It follows trivially from Lemma 5 that $|\mathcal{L}_H| \leq 2|\mathcal{L}_{max}|$. Consequently, the total number of iterations through this loop during the course of the algorithm is $O(m)$. Efficient implementation of the sets $\mathcal{P}$ and $\mathcal{C}(P)$ in lines 14–17 is quite easy. An array of markers can be used to detect membership in $\mathcal{P}$ in constant time (see line 15). In addition, the set $\mathcal{P}$ should be implemented as a doubly-linked list to enable insertion and deletion of members in constant time (see lines 10, 14, 15, 23). The sets $\mathcal{C}(P)$ can be implemented as singly-linked lists (see lines 7, 16, 17, 24, 27), with a pointer to the tail of each list to implement the set union in line 16 in constant time. Thus, the total work associated with the lines of the loop not excluded from consideration is $O(m)$.

Consider now the **for** loop in lines 23–30. The total work associated with lines 24–26 is $O(m)$. Since $|\mathcal{P}| \leq |\mathcal{L}_{max}|$ and each of the lines 27–29 is a constant-time operation, the total work required by this loop is $O(m)$.

Finally, we consider the lines of the algorithm associated with accessing or changing the clique tree used to represent the current reduced chordal graph (lines 1, 13, 18, and 19). The key lines to be discussed are 13, 18, and 19.

The data structure used to represent clique trees in [23] and the techniques used to eliminate cliques from it are sufficient to implement these three lines efficiently. The data structure is a *rooted* clique tree with the nodes of each clique listed in ascending order by some perfect elimination ordering of the underlying chordal graph. (If necessary, such an ordering can be obtained from the maximum cardinality search algorithm used to construct the input.) The data structure initially has the children of each parent listed in descending order by the size of the separator each shares with the parent. Sorting the nodes of the cliques can be done in $O(q)$ time, and sorting the children in their lists can be done in $O(n)$ time, both using a bucket sort. As the algorithm proceeds, careful maintenance of a partial ordering in the lists of children enables selection of P in line 13 after inspecting at most two neighbors of K in T', namely the parent of K and the first clique in its list of children. By thus avoiding a search among all neighbors of K, line 13 becomes a constant time operation.

Computing $T' \setminus_r \{K\}$ in line 18 and updating the parameters in line 19 requires a few simple operations for each node of $Sep_H(K)$ and other work of lower order time complexity. Consequently the total work done in lines 18 and 19 is $O(q)$. The details can be found in [23].

From the above discussion and the fact that $m \leq n$, the algorithm has $O(n+q)$ total time complexity. This, together with the time required to obtain the input, gives us an $O(n+e)$ time algorithm.

6. Concluding remarks

The primary contribution of this paper is an efficient algorithm for generating a minimum-diameter clique tree, along with an analysis of its time complexity. The algorithm is a natural generalization of the obvious greedy algorithm for rooting an ordinary tree in order to minimize its height, and can be viewed as a block variant of the Jess and Kees ordering algorithm [23, 25]. To achieve this generalization, we defined the leaf set $\mathcal{L}_G$ to include every clique that is a leaf in some clique tree in $\mathcal{T}_G$. We then introduced characterizations of the cliques in $\mathcal{L}_G$ that help to compute the set very efficiently. This was followed by a characterization of maximum cardinality leaf sets. We then presented the obvious greedy algorithm, which repeats the following major step until the graph has been eliminated: compute a maximum cardinality leaf set, eliminate these leaf cliques from the graph, and collect an appropriate set of clique tree edges incident upon these leaves. We then showed that this algorithm generates a minimum-diameter clique tree.

To demonstrate that the new algorithm executes in $O(n + e)$ time, we addressed several implementation issues, the most important of which is efficient computation of the maximum cardinality leaf sets. An actual code based on the detailed algorithm would maintain a clique tree representation of the *current* chordal graph that may not have minimum diameter. Lewis et al. [23] contains details about the data structure used to store this sequence of clique trees and how they are used to implement the elimination process very efficiently.

We believe that our algorithm will be useful in a number of application areas. Of particular interest to us is its use in an efficient implementation of a parallel sparse Cholesky factorization algorithm and also an efficient parallel method for calculating probability distributions in a probabilistic knowledge-based system. The next two paragraphs briefly discuss the application of our results in these two areas.

Gilbert and Schreiber [15] have recently implemented a fine-grained parallel sparse Cholesky algorithm on the Connection Machine, a massively-parallel distributed-memory SIMD machine (Single-Instruction-Multiple-Data). Their algorithm is a highly parallel variant of the multifrontal method for sparse factorization [7, 24]. To improve performance they use an elimination sequence obtained by repeating the following step until all nodes have been eliminated: remove all simplicial nodes from the current chordal graph. Our results can be used to demonstrate that the number of major steps taken by their ordering algorithm, and consequently their factorization algorithm, is the minimum possible. This is of practical importance because between each major step (and only then) their factorization algorithm must issue calls to the Connection Machine's general router to accumulate results and communicate them from one processor to another to set up the next major step. Calls to the general router are so expensive that the height of the clique tree, though not the dominant time-complexity term in a theoretical sense, is nonetheless dominant in the practical sense. Their ordering algorithm is based on this assessment, and the analysis in this paper can be used to demonstrate that they have minimized the number of calls to the router. In addition, the results in this paper possibly provide a basis for reorganizing their factorization algorithm to improve its efficiency; however, further study will be required to determine if substantial improvements are indeed possible.

Lauritzen and Spiegelhalter [22] have presented a technique for calculating probability distributions in knowledge-based systems in which probabilities of discrete-valued random variables are an inherent component of the encoded knowledge. Briefly, a probabilistic knowledge-based system is a *Markov network* $M = (V, E_M, Pr)$. (M is a digraph with nodes V being the system random variables, directed arcs E_M taken from $V \times V$, and probability distributions Pr corresponding to the acyclic arc-structure.) The goal is to maintain the probability distributions Pr as they vary with time and queries of the network. To achieve this, the directed graph M is first converted into the corresponding undirected graph G, then edges are added as needed to convert G into a chordal graph. The probability distributions can be maintained with added efficiency by using a clique tree representation of G to organize the computation. Backward and forward propagation of data in the clique tree, which in practice may require the manipulation of large tables of probabilities, is a fundamental part of the method. England et al. [9, 10] describe aspects of the Pr component of M that render certain sections of the data propagation computationally independent. This data independence can be exploited not only to avoid unnecessary computations in a conventional sequential implementation, but also to allow simultaneous execution within as many cliques as possible in a parallel implementation. To complement these results and allow for an even greater amount of parallelism in the solution process, it may be advantageous to further exploit the structure of the underlying graph of M. One way to do this is to use a clique tree representation of minimum diameter.

There are several open questions worth mentioning. In light of the algorithm's possible applications, it is worthwhile to consider how to implement it (or some variant thereof) to run efficiently on a parallel machine, particularly a fined-grained machine such as the Connection Machine. Our algorithm finds a maximum-weight, minimum-height spanning tree of the clique intersection graph of a given chordal graph. Camerini et al. [5] have shown that for general weighted graphs this problem is NP-complete. It would be interesting to know whether or not a *maximum*-diameter clique tree (or equivalently a maximum-weight, maximum-height spanning tree of the clique intersection graph of G) can be found in polynomial time.

Acknowledgments. The authors would like to thank Eduardo D'Azevedo, John Gilbert, Eric Kirsch, and Esmond Ng for many valuable comments and suggestions.

7. 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-73-88, Department of Computer Science, The University of Tennessee, Knoxville, Tennessee, 1988.

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

[5] P.M. Camerini, G. Galbiati, and F. Maffioli. Complexity of spanning tree problems: Part I. *European Journal of Operational Research*, 5:346–352, 1980.

[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. The multifrontal solution of indefinite sparse symmetric systems of equations. *ACM Trans. Math. Software*, 9:302–325, 1983.

[8] 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.

[9] R.E. England. *Clique graph models for independent computations*. PhD thesis, Dept. of Computer Science, The University of Tennessee, 1989.

[10] R.E. England, J.R.S. Blair, and M.G. Thomason. Independent computations in a probablistic knowledge-based system. Technical Report CS-90-128, Department of Computer Science, The University of Tennessee, Knoxville, Tennessee, 1991.

[11] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. Assoc. Comput. Mach.*, 30:514–550, 1983.

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

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

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

[15] J.R. Gilbert and R. Schreiber. Highly parallel sparse Cholesky factorization. Technical Report CSL 90–7, Xerox Palo Alto Research Center, 1990. (submitted to *SIAM J. Sci. Statist. Comp.*).

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

[17] C-W. Ho and R. C. T. Lee. Efficient parallel algorithms for finding maximal cliques, clique trees, and minimum coloring on chordal graphs. *Inform. Process. Lett.*, 28:301–309, 1988.

[18] 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.

[19] F.V. Jensen. Junction trees and decomposable hypergraphs. Technical report, JUDEX, Aalborg, Denmark, 1988.

[20] 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.

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

[22] S.L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their applications to expert systems. *J. Royal Statist. Soc., ser B*, 50:157–224, 1988.

[23] 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.

[24] J. W-H. Liu. The multifrontal method for sparse matrix solution: theory and practice. Technical Report CS-90-04, Department of Computer Science, York University, North York, Ontario, Canada, 1990.

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

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

[28] 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.

[29] R.E. Tarjan. *Data Structures and Network Algorithms*. SIAM, Philidelphia, 1983.

[30] 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.

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

A. Notation

For easy reference we have included the following table of informal definitions for most of the notation introduced in the paper.

$G = (V, E)$	–	A chordal graph (G, G', H and H').
$T = (\boldsymbol{\mathcal{K}}_G, \boldsymbol{\mathcal{E}})$	–	A clique tree (T, T', T_{mst}, T_{ct}, T_{alg}, and T_{min}).
$\boldsymbol{\mathcal{K}}_G$	–	The maximal cliques of G ($\boldsymbol{\mathcal{K}}_G$, $\boldsymbol{\mathcal{K}}_{G'}$, $\boldsymbol{\mathcal{K}}_H$, and $\boldsymbol{\mathcal{K}}_{H'}$).
$\boldsymbol{\mathcal{K}}(S)$	–	The cliques containing $S \subseteq V$ ($\boldsymbol{\mathcal{K}}(S)$, $\boldsymbol{\mathcal{K}}_H(S)$, $\boldsymbol{\mathcal{K}}_H(S')$, and $\boldsymbol{\mathcal{K}}_{H'}(S)$).
$\boldsymbol{\mathcal{T}}_G$	–	The clique trees of G ($\boldsymbol{\mathcal{T}}_G$, $\boldsymbol{\mathcal{T}}_{G'}$, $\boldsymbol{\mathcal{T}}_H$, and $\boldsymbol{\mathcal{T}}_{H'}$).
$\boldsymbol{\mathcal{E}}$	–	The edges of a clique tree ($\boldsymbol{\mathcal{E}}$, $\boldsymbol{\mathcal{E}}_{T'}$, and $\boldsymbol{\mathcal{E}}_{min}$).
$\boldsymbol{\mathcal{M}}_T$	–	The *multiset* of separators of T ($\boldsymbol{\mathcal{M}}_T$ and $\boldsymbol{\mathcal{M}}_{T'}$).
$\boldsymbol{\mathcal{M}}_G$	–	The *multiset* of separators of G ($\boldsymbol{\mathcal{M}}_G$, $\boldsymbol{\mathcal{M}}_{G'}$, $\boldsymbol{\mathcal{M}}_H$, and $\boldsymbol{\mathcal{M}}_{H'}$).
$\boldsymbol{\mathcal{S}}(K)$	–	The *set* of separators included in clique K ($\boldsymbol{\mathcal{S}}(K)$ and $\boldsymbol{\mathcal{S}}_H(K)$).
$\overline{\boldsymbol{\mathcal{S}}}(K)$	–	The *set* of maximal separators among those in clique K ($\overline{\boldsymbol{\mathcal{S}}}(K)$ and $\overline{\boldsymbol{\mathcal{S}}}_H(K)$).
$\overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_G)$	–	The *set* of leaf separators ($\overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_G)$ and $\overline{\boldsymbol{\mathcal{S}}}(\boldsymbol{\mathcal{L}}_H)$).
$\boldsymbol{\mathcal{L}}_T$	–	The leaves of T ($\boldsymbol{\mathcal{L}}_T$, $\boldsymbol{\mathcal{L}}_{T'}$, $\boldsymbol{\mathcal{L}}_{T_{alg}}$, and $\boldsymbol{\mathcal{L}}_{T_{min}}$).
$\boldsymbol{\mathcal{L}}_G$	–	The leaf cliques of G ($\boldsymbol{\mathcal{L}}_G$, $\boldsymbol{\mathcal{L}}_{G'}$, $\boldsymbol{\mathcal{L}}_H$, and $\boldsymbol{\mathcal{L}}_{H'}$).
$\boldsymbol{\mathcal{L}}_{max}$	–	A maximum cardinality set of leaf cliques.
$\boldsymbol{\mathcal{L}}(S)$	–	The leaves $K \in \boldsymbol{\mathcal{L}}_G$ for which $\overline{\boldsymbol{\mathcal{S}}}(K) = \{S\}$ ($\boldsymbol{\mathcal{L}}(S)$, $\boldsymbol{\mathcal{L}}(S')$, $\boldsymbol{\mathcal{L}}_H(S)$, $\boldsymbol{\mathcal{L}}_H(S')$, and $\boldsymbol{\mathcal{L}}_H(S'')$).
$-$	–	Set difference operator.
$\setminus$	–	$G \setminus A$ is the subgraph of G induced by $V - A$.
$\setminus_r$	–	$T \setminus_r \{K\}$ is the removal of $K \in \boldsymbol{\mathcal{L}}_G$ from T, with restructuring when $K \notin \boldsymbol{\mathcal{L}}_T$.
$\sigma(K)$	–	The size of the largest separator in $\overline{\boldsymbol{\mathcal{S}}}(K)$ ($\sigma(K)$, $\sigma_G(K)$, $\sigma_H(K)$, and $\sigma_{H'}(K)$).
$\rho(K)$	–	The number of simplicial nodes in clique K ($\rho(K)$, $\rho_G(K)$, $\rho_H(K)$, and $\rho_{H'}(K)$).