Nordic Journal of Computing 1(1994), 173–201.

ON FINDING MINIMUM-DIAMETER CLIQUE TREES *

JEAN R. S. BLAIR
Department of Electrical Engineering and Computer Science
United States Military Academy
West Point, NY 10996-5000
U.S.A.
`blair@eecs1.eecs.usma.edu`

BARRY W. PEYTON
Mathematical Sciences Section
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367
U.S.A.
`peyton@msr.epm.ornl.gov`

Abstract. 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. 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.

ACM CCS Categories and Subject Descriptors: F.2.2, G.2.2

Key words: chordal graphs, clique trees, acyclic hypergraphs, parallel computing

1. Introduction

Chordal graphs arise in several application areas including data-base management systems [1, 10, 25], knowledge-based systems [9, 17, 18], and the solution of sparse symmetric linear systems of equations [14, 19, 21, 22, 23]. 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 [18, 19, 25]. 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 [13, 16] have shown that a tight upper bound on the number of distinct clique trees is exponential in the number of nodes in the graph. It is interesting from a

*Research supported by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc.

Received November 1993. Accepted March 1994.

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) is a natural candidate. In particular this is the case when the parallel algorithm in question has a leading 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.

This paper introduces a linear-time algorithm for computing a minimum-diameter clique tree. The essential character of the algorithm 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) in the current tree, and eliminate each of these nodes and the single edge incident on 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.

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. Section 5 presents a detailed version of our algorithm and presents other material needed to verify the linear time-complexity of the algorithm. The concluding remarks in Section 6 include a brief discussion of the two application areas where a minimum-diameter clique tree should prove useful.

2. Clique trees: background

We assume the reader is familiar with standard graph terminology (see, for example, Golumbic [15]). 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. Each item in our notation will use (as needed) a subscript to identify which chordal graph or clique tree it pertains to. This subscript is suppressed where the relevant graph is known by context.

2.1 Definition of clique trees

A graph is *chordal* (triangulated, rigid circuit) if every cycle of length ≥ 4 contains a *chord*, i.e., an edge joining two non-adjacent nodes in the cycle. Let $G = (V, E)$ be a chordal graph, and let $\mathcal{K}_G = \{K_1, K_2, \ldots, K_m\}$ be the set containing the maximal cliques in G. Throughout this paper the term *clique* always refers to a maximal clique, and the term *maximal clique* is used only where emphasis on maximality seems warranted. The graph G is assumed to be connected; that all definitions and results generalize to disconnected chordal graphs should be readily apparent.

Various characterizations of clique trees (also called acyclic hypergraphs or join trees) have appeared in the literature [1, 2, 4, 12, 24, 26]. We define clique trees here using the following theorem, which is easily derived from a more general result due to Buneman [4], Gavril [12], and Walter [26].

THEOREM 1. *A graph $G = (V, E)$ is chordal if and only if there exists a tree $T = (\mathcal{K}_G, \mathcal{E})$ that satisfies the following property: for every pair of distinct cliques $K, K' \in \mathcal{K}_G$, the intersection $K \cap K'$ is contained in every clique on the path connecting K and K' in T.*

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

The reader may verify that the tree in Figure 2 is a clique tree of the chordal graph shown in Figure 1. The graph in Figure 1 will be used throughout this paper to illustrate results and key points. For convenience we shall refer to the nodes of this graph as $v_1, v_2, \ldots, v_{10}$; e.g., the node labeled "6" will be referred to as v_6.

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 $\mathcal{K}_G$. Two distinct cliques K and K' are joined by an edge if and only if their intersection is nonempty; moreover, each such edge $\{K, K'\}$ is assigned a positive weight given by $|K \cap K'|$. Bernstein, Goodman, and Gavril [2, 13] have shown that, for any chordal graph G, the set of clique trees $\mathcal{T}_G$ is precisely the set of maximum-weight spanning trees of the clique-intersection graph associated with G.

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

The reader may verify that the weighted graph shown in Figure 3 is the clique-intersection graph of the graph shown in Figure 1, and that the clique tree shown in Figure 2 is a maximum-weight spanning tree of the clique-intersection graph.

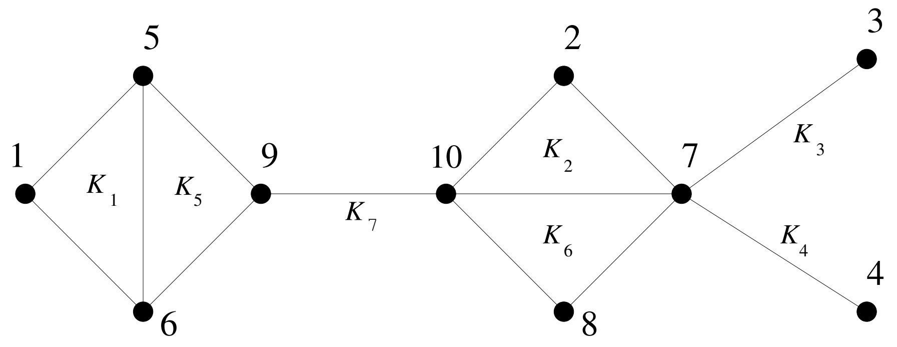


Fig. 1: A chordal graph.

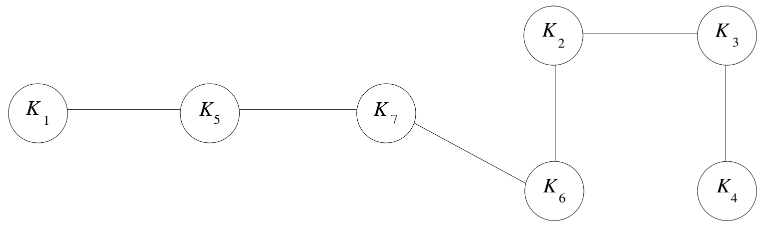


Fig. 2: A clique tree of the graph in Figure 1.

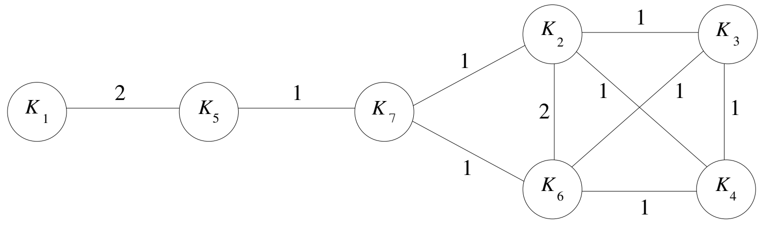


Fig. 3: The clique-intersection graph associated with the graph in Figure 1. Each edge weight appears beside the edge to which it is assigned.

2.2 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 distinct connected components. If no proper subset of S has this property, then S is said to be a *minimal a-b separator*. When the pair of nodes remains unspecified, we call S a *minimal node-pair separator*.

For any clique tree $T = (\mathcal{K}_G, \mathcal{E}) \in \mathcal{T}_G$, consider the *multiset* given by

$$\mathcal{M}_T := \Big\{ K \cap K' \;\Big|\; \{K, K'\} \in \mathcal{E} \Big\}.$$

Ho and Lee [16] showed that every member of $\mathcal{M}_T$ is a minimal node-pair separator; they further showed that for each minimal node-pair separator S of G, $\mathcal{M}_T$ contains a number of copies of S that is invariant over all clique trees $T \in \mathcal{T}_G$. Consequently, we have $\mathcal{M}_T = \mathcal{M}_{T'}$ for all $T, T' \in \mathcal{T}_G$. For brevity, we will refer to each member of $\mathcal{M}_T$ as a *separator*. Henceforth, let $\mathcal{M}_G$ denote the *multiset of separators* associated with each clique tree in $\mathcal{T}_G$. For the graph in Figure 1, we have $\mathcal{M}_G = \{\{v_5, v_6\}, \{v_9\}, \{v_{10}\}, \{v_7, v_{10}\}, \{v_7\}, \{v_7\}\}$.

For any set of nodes $S \subseteq V$, the *set of cliques containing* S, denoted $\mathcal{K}(S)$, is given by

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

In this paper the set S will always be a separator taken from $\mathcal{M}_G$. It is worth emphasizing that every separator $S \in \mathcal{M}_G$ is contained in at least two cliques [i.e., $|\mathcal{K}(S)| \geq 2$].

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

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

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

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

Consider again the graph shown in Figure 1. Table I shows the sets $\mathcal{S}(K)$ and $\overline{\mathcal{S}}(K)$ for each clique in the graph.

Clique K	$\mathcal{S}(K)$	$\overline{\mathcal{S}}(K)$
$K_1 = \{v_1, v_5, v_6\}$	$\{v_5, v_6\}$	$\{v_5, v_6\}$
$K_2 = \{v_2, v_7, v_{10}\}$	$\{v_7\}, \{v_7, v_{10}\}, \{v_{10}\}$	$\{v_7, v_{10}\}$
$K_3 = \{v_3, v_7\}$	$\{v_7\}$	$\{v_7\}$
$K_4 = \{v_4, v_7\}$	$\{v_7\}$	$\{v_7\}$
$K_5 = \{v_5, v_6, v_9\}$	$\{v_5, v_6\}, \{v_9\}$	$\{v_5, v_6\}, \{v_9\}$
$K_6 = \{v_7, v_8, v_{10}\}$	$\{v_7\}, \{v_7, v_{10}\}, \{v_{10}\}$	$\{v_7, v_{10}\}$
$K_7 = \{v_9, v_{10}\}$	$\{v_9\}, \{v_{10}\}$	$\{v_9\}, \{v_{10}\}$

TABLE I: Sets of separators for each clique in the graph shown in Figure 1.

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

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

PROOF. Choose a separator $S \in \overline{\mathcal{S}}(K)$, and choose $P \in \mathcal{K}(S) - \{K\}$. (Throughout this paper the binary set difference operator is "$-$".) Consider the path $K = K_1, K_2, \ldots, K_r = P$ from K to P in T. It follows from Theorem 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 $\mathcal{S}(K)$ we have $K \cap K_2 = S$, which proves the result. □

3. Leaf cliques

For any clique tree $T \in \mathcal{T}_G$, let $\mathcal{L}_T$ be the set containing the leaves in T (i.e., the members of $\mathcal{K}_G$ with degree one in T). We then let $\mathcal{L}_G$, the *leaf cliques in* G, be the set containing every clique that is a leaf in at least one clique tree $T \in \mathcal{T}_G$. Section 3.1 contains a simple characterization of $\mathcal{L}_G$. With $\mathcal{L}_G$ in hand, the minimum-diameter clique tree algorithm must then compute a set of leaf cliques $\mathcal{L}_{\max} \subseteq \mathcal{L}_G$ such that $\mathcal{L}_{\max} = \mathcal{L}_T$ for a clique tree $T \in \mathcal{T}_G$, and moreover $|\mathcal{L}_T| \geq |\mathcal{L}_{T'}|$ for every clique tree $T' \in \mathcal{T}_G$ (see Section 4.2 for proof). Section 3.2 contains a characterization of these *maximum-cardinality leaf sets* $\mathcal{L}_{\max} \subseteq \mathcal{L}_G$.

3.1 A characterization of leaf cliques

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

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

PROOF. Let S be the sole member of $\overline{\mathcal{S}}(K)$, and suppose that K is *not* a leaf in $T \in \mathcal{T}_G$ [see Figure 4(a)]. Choose $P \in \mathcal{K}(S) - \{K\}$. It follows from Theorem 1 that $S \subset P'$, where P' is the clique adjacent to K on the path from K to P in T (possibly $P' = P$). Consider a clique $C \neq P'$ that is also adjacent to K in T. By Theorem 1, $C \cap P \subseteq C \cap K$. Furthermore, since S is the only member of $\overline{\mathcal{S}}(K)$, we have $C \cap K \subseteq S \subset P$; whence $C \cap K \subseteq C \cap P$. It follows that $C \cap K = C \cap P$, and hence the edges $\{C, K\}$ and $\{C, P\}$ have the same weight. Thus, by Theorem 2, the tree obtained from T by removing the edge $\{C, K\}$ and adding the edge $\{C, P\}$ is also a clique tree. Repeating this process for every clique $C_i \neq P'$ that is adjacent to K in T, we obtain a new clique tree T' in which K is a leaf [see Figure 4(b)], and 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

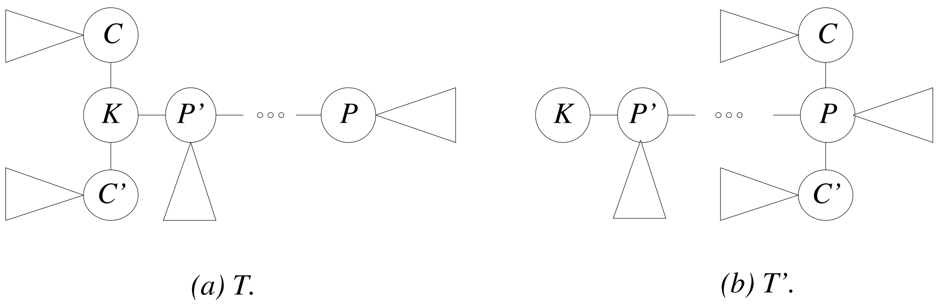


Fig. 4: Transformation of T into T' in which K is a leaf, as discussed in the proof of Lemma 2.

in subsequent proofs. We note here that the parameters required for this operation are a clique tree T, a leaf clique $K \in \mathcal{L}_G - \mathcal{L}_T$ and any clique $P \neq K$ that contains the sole separator S found in $\overline{\mathcal{S}}(K)$. Whenever 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 by Theorem 1, $S \subset P'$, clearly P' can play the role of P, as will be the case in a key application of this operation in Section 4.

The next lemma completes the characterization of $\mathcal{L}_G$.

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

Proof. Sufficiency for membership in $\mathcal{L}_G$ follows immediately from Lemma 2. To prove necessity, choose $K \in \mathcal{L}_G$ and let $T \in \mathcal{T}_G$ 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 $\overline{\mathcal{S}}(K)$. $\square$

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 in a chordal graph. That is, a clique K is a leaf clique in G if it has only one member of $\overline{\mathcal{S}}(K)$ through which it can be joined to neighbors in a clique tree. Applying Lemma 3 to the example (refer to the last column in Table I), we see that $\mathcal{L}_G = \{K_1, K_2, K_3, K_4, K_6\}$.

3.2 Maximum-cardinality leaf sets

In this section we give a useful characterization of the leaf sets $\mathcal{L}_T$ for which $T \in \mathcal{T}_G$ and $|\mathcal{L}_T| \geq |\mathcal{L}_{T'}|$ for every $T' \in \mathcal{T}_G$. We have shown in Lemma 3 that each leaf clique $K \in \mathcal{L}_G$ has associated with it a unique separator $S \in \mathcal{M}_G$ that contains every separator that lies within K. For every such *leaf separator* S, let $\mathcal{L}(S)$ be the subset of $\mathcal{L}_G$ given by

$$\mathcal{L}(S) := \left\{ K \in \mathcal{L}_G \,\middle|\, \overline{\mathcal{S}}(K) = \{S\} \right\}.$$

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

Remark 1. Each leaf belongs to precisely one leaf-cohort set, and therefore the collection of leaf-cohort sets forms a partition of $\mathcal{L}_G$.

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

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

To prove the converse assume that $\mathcal{K}(S) - \mathcal{L}(S) \neq \emptyset$, and let $P \in \mathcal{K}(S) - \mathcal{L}(S)$. Let $T \in \mathcal{T}_G$, and suppose that there exists a clique $K \in \mathcal{L}(S) - \mathcal{L}_T$. As in the proof of Lemma 2 (see Figure 4), 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 the desired result. □

Lemma 5. *Assume $|\mathcal{K}_G| \geq 3$ and $T \in \mathcal{T}_G$. Then $|\mathcal{L}_T| \geq |\mathcal{L}_{T'}|$ for every $T' \in \mathcal{T}_G$ if and only if for each leaf separator S:*

(1) if $\mathcal{L}(S) \subset \mathcal{K}(S)$ then $\mathcal{L}(S) \subseteq \mathcal{L}_T$.

(2) if $\mathcal{L}(S) = \mathcal{K}(S)$ then $|\mathcal{L}(S) - \mathcal{L}_T| = 1$.

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 possesses 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 for which $\mathcal{L}(S) \subset \mathcal{K}(S)$ we have a clique $K \in \mathcal{L}(S)$ that is not a leaf in 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 4), 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 in T remain leaves in T', it follows that T' has one more leaf than T. This suffices to show that 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 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 in T. Let K and P be two such cliques. From this point the argument runs the same course as the argument in the previous paragraph, completing the proof. □

Lemma 5 and Remark 1 characterize a maximum-cardinality leaf set $\mathcal{L}_{\max}$: $\mathcal{L}_{\max}$ includes the elements of $\mathcal{L}(S)$ for every $\mathcal{L}(S) \subset \mathcal{K}(S)$, and includes all but one (any one) of the elements of $\mathcal{L}(S)$ for every $\mathcal{L}(S) = \mathcal{K}(S)$.

Returning to our example, we have three leaf separators: $\{v_5, v_6\}$, $\{v_7, v_{10}\}$, and $\{v_7\}$. Table II shows $\mathcal{L}(S)$ and $\mathcal{K}(S)$ for each of the leaf separators. Note that by Lemma 5 any $\mathcal{L}_{\max}$ for the example graph must contain K_1, exactly one of K_2 and K_6, and both K_3 and K_4.

Leaf separator S	$\mathcal{L}(S)$	$\mathcal{K}(S)$
$S_1 = \{v_5, v_6\}$	K_1	K_1, K_5
$S_2 = \{v_7, v_{10}\}$	K_2, K_6	K_2, K_6
$S_3 = \{v_7\}$	K_3, K_4	K_2, K_3, K_4, K_6

TABLE II: Cohorts of leaf cliques for each leaf separator in the graph in Figure 1.

4. The 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. Section 4.1 gives a high-level description of the new algorithm and proves that it generates a clique tree. Section 4.2 proves that the resulting clique tree has minimum diameter.

4.1 Computing the clique tree

A *simplicial node* in G is any node whose adjacency set is a clique in G. It is trivial to show that a node is simplicial if and only if it belongs to precisely one maximal clique, and we will make extensive use of this observation throughout the remainder of the paper. It is well known that any chordal graph can be reduced to the null graph by successive removal of simplicial nodes [6, 11, 15, 23]. The order in which the simplicial nodes are removed is known as a *perfect elimination ordering*. Our algorithm eliminates the nodes of the graph in a perfect elimination ordering; more specifically, each major step eliminates the cliques belonging to a maximum-cardinality leaf set by removing from the graph all simplicial nodes lying in these cliques.

A node $v \in K$ belongs to two or more cliques in G if and only if it belongs to a separator in $\mathcal{S}(K)$. It follows from Lemma 3 and the preceding observation that any leaf clique $K \in \mathcal{L}_G$ can be partitioned into two sets

$$K = \mathrm{Sim}(K) \cup \mathrm{Sep}(K),$$

where $\mathrm{Sim}(K)$ contains the simplicial nodes in K and $\mathrm{Sep}(K)$ contains the nodes that constitute the leaf separator associated with K. For $K \in \mathcal{L}_G$, let $G \setminus \mathrm{Sim}(K)$ denote the graph obtained by eliminating from G the simplicial nodes in $\mathrm{Sim}(K)$ and their incident edges. In other words, $G \setminus \mathrm{Sim}(K)$ is the subgraph of G induced by $V - \mathrm{Sim}(K)$. Since $\mathrm{Sep}(K)$, which contains the nodes of K remaining in $G \setminus \mathrm{Sim}(K)$, is contained in at least one other maximal clique P of G, it follows that K disappears from $G \setminus \mathrm{Sim}(K)$ as a maximal clique, and can be viewed as “absorbed” by P. Moreover, since the nodes in $\mathrm{Sim}(K)$ belong to no other maximal clique in G, the other maximal cliques in G remain unchanged in $G \setminus \mathrm{Sim}(K)$. Thus, $G \setminus \mathrm{Sim}(K)$ is precisely the chordal graph whose set of maximal cliques is given by $\mathcal{K}_G - \{K\}$.

Our algorithm for computing a minimum-diameter clique tree is shown in Figure 5. At the beginning of each major step (i.e., each iteration of the

```
$\mathcal{E}_{\min} \leftarrow \emptyset$ ;
$H \leftarrow G$ ; $H' \leftarrow G$ ;
**while** $|\mathcal{K}_H| \geq 3$ **do**
    Choose a maximum-cardinality leaf set $\mathcal{L}_{\max} \subseteq \mathcal{L}_H$ ;
    **for** $K \in \mathcal{L}_{\max}$ **do**
        Choose $P \in \mathcal{K}_H - \mathcal{L}_{\max}$ for which $\mathrm{Sep}_H(K) \subset P$ ;
        $\mathcal{E}_{\min} \leftarrow \mathcal{E}_{\min} \cup \{K, P\}$ ;
        $H' \leftarrow H' \setminus \mathrm{Sim}_H(K)$ ;
    **end for** ;
    $H \leftarrow H'$ ;
**end while** ;
**if** $|\mathcal{K}_H| = 2$ **do**
    $\mathcal{E}_{\min} \leftarrow \mathcal{E}_{\min} \cup \{K, P\}$, where $\{K, P\} = \mathcal{K}_H$ ;
**end if** ;
```

Fig. 5: Algorithm for generating a minimum-diameter clique tree.

while loop), H is the remaining chordal graph from which a maximum-cardinality leaf set $\mathcal{L}_{\max}$ will be removed. After $\mathcal{L}_{\max}$ has been selected, the **for** loop processes the leaf cliques one at a time. For each leaf clique in $\mathcal{L}_{\max}$ it finds a parent clique and eliminates the clique from the graph.

As an illustration, consider again the graph shown in Figure 1. Let $\mathcal{L}_{\max} = \{K_1, K_2, K_3, K_4\}$ in the first iteration through the **while** loop. Then the only possible parent for K_1 is K_5, because K_5 is the only clique in $\mathcal{K}_H - \mathcal{L}_{\max}$ that contains K_1's leaf separator. Similarly, K_6 must be the parent of K_2, K_3, and K_4 (see Figure 6). Eliminating the simplicial nodes from the

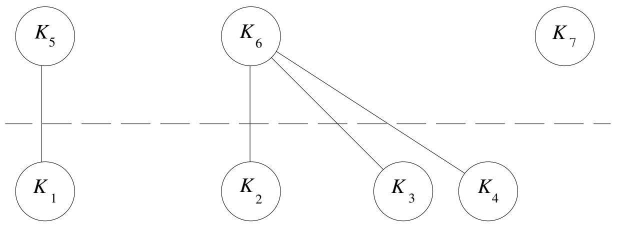


Fig. 6: Partially constructed clique tree of the graph in Figure 1, after the first major step of the algorithm in Figure 5. Cliques shown below the dotted line are eliminated from the graph; cliques shown above the dotted line remain in the reduced graph.

leaf cliques in $\mathcal{L}_{\max}$ results in the reduced graph shown in Figure 7, with separators and leaf cohort sets shown in the tables. The second iteration through the **while** loop completes the algorithm by choosing K_7 as the parent to both leaf cliques of the reduced graph (see Figure 8).

Recall that for each major step in the algorithm, the elimination of the leaf set $\mathcal{L}_{\max}$ involves the elimination of the members of $\mathcal{L}_{\max}$ one at a time in some arbitrary order. Clearly, this approach is based on the assumption that elimination of $K \in \mathcal{L}_{\max}$ by an iteration of the **for** loop causes none of the uneliminated cliques in $\mathcal{L}_{\max}$ to become a non-leaf in the reduced graph. The following simple lemma will be used to address this and other closely related issues associated with our algorithm.

Lemma 6. *Assume $|\mathcal{K}_G| \geq 3$. Let $T \in \mathcal{T}_H$ and $K \in \mathcal{L}_T$, and consider $T' = T \setminus \{K\}$ and $H' = H \setminus \mathrm{Sim}_H(K)$. The following properties hold for H, T, H' and T':*

(1) $T' \in \mathcal{T}_{H'}$.

(2) $\mathcal{L}_T - \{K\} \subseteq \mathcal{L}_{T'} \subseteq \mathcal{L}_{H'}$.

(3) For each $K' \in \mathcal{L}_T - \{K\}$, we have $\mathrm{Sim}_{H'}(K') = \mathrm{Sim}_H(K')$ and $\mathrm{Sep}_{H'}(K') = \mathrm{Sep}_H(K')$.

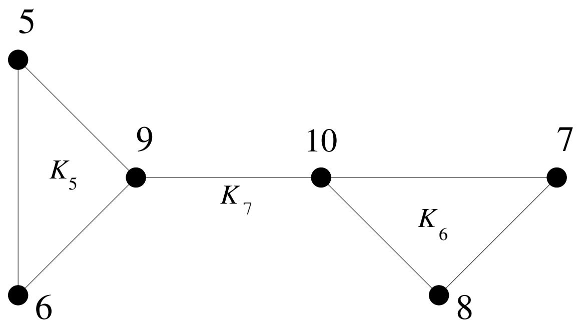
Proof. Let $K', K'' \in \mathcal{K}_{H'} = \mathcal{K}_H - \{K\}$. From the definition of T and T', it follows that the path connecting K' and K'' in T' is identical to the one connecting the pair in T. Thus, by Theorem 1 (applied to $T \in \mathcal{T}_H$) we have $T' \in \mathcal{T}_{H'}$.

Clearly, any leaf in $\mathcal{L}_T - \{K\}$ remains a leaf in T', and thus $\mathcal{L}_T - \{K\} \subseteq \mathcal{L}_{T'} \subseteq \mathcal{L}_{H'}$.

Choose $K' \in \mathcal{L}_T - \{K\}$ and let $\{K', P\}$ be the single edge incident on K' in T. Since $|\mathcal{K}_G| \geq 3$, we know that $P \neq K$. Since $\{K', P\}$ is the single edge incident on K' in T', it follows from Lemma 1 that $K' \cap P = \mathrm{Sep}_{H'}(K') = \mathrm{Sep}_H(K')$. Furthermore, since K' is a leaf clique in both H and H', we have

$$\mathrm{Sep}_{H'}(K') \cup \mathrm{Sim}_{H'}(K') = K' = \mathrm{Sep}_H(K') \cup \mathrm{Sim}_H(K'),$$

whence $\mathrm{Sim}_{H'}(K') = \mathrm{Sim}_H(K')$. $\square$



Separators within each clique

Clique K	$\mathcal{S}(K)$	$\overline{\mathcal{S}}(K)$
$K_5 = \{v_5, v_6, v_9\}$	$\{v_9\}$	$\{v_9\}$
$K_6 = \{v_7, v_8, v_{10}\}$	$\{v_{10}\}$	$\{v_{10}\}$
$K_7 = \{v_9, v_{10}\}$	$\{v_9\}, \{v_{10}\}$	$\{v_9\}, \{v_{10}\}$

Cohorts clustered around leaf separators

Leaf separator S	$\mathcal{L}(S)$	$\mathcal{K}(S)$
$S_4 = \{v_9\}$	K_5	K_5, K_7
$S_5 = \{v_{10}\}$	K_6	K_6, K_7

Fig. 7: The reduced graph after the first major step of the algorithm in Figure 5. Sets of separators for each clique and cohorts of leaf cliques for each leaf separator are given in the tables.

Let $\mathcal{L}_{\max} \subseteq \mathcal{L}_H$ be the leaf set chosen for elimination during a major step of the algorithm in Figure 5. Applying Lemmas 5 and 6, we justify several details found in the inner loop with the following remarks.

Remark 2. The algorithm assumes the existence of an appropriate "parent" $P \in \mathcal{K}_H - \mathcal{L}_{\max}$ for each leaf $K \in \mathcal{L}_{\max}$; Lemma 5 ensures the existence of such a clique.

Remark 3. Select a clique tree $T \in \mathcal{T}_H$ for which $\mathcal{L}_T$ is identical to the maximum-cardinality leaf set $\mathcal{L}_{\max}$ chosen by the algorithm. Repeated application of Properties 1 and 2 of Lemma 6 ensure that after the removal of a leaf clique in the inner loop, the uneliminated members of $\mathcal{L}_{\max}$ remain leaves in the reduced graph, as required during subsequent iterations of the inner loop.

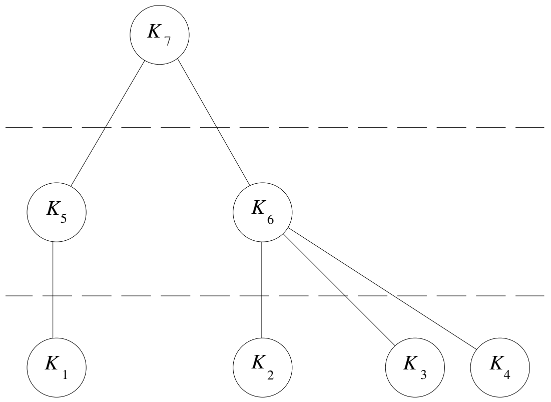


Fig. 8: The clique tree produced by the algorithm in Figure 5 applied to the graph in Figure 1. Dotted lines separate the maximum cardinality leaf sets chosen in each of the two iterations through the **while** loop.

Remark 4. Similarly, repeated application of Properties 1 and 3 of Lemma 6 ensure that $\mathrm{Sep}_{H'}(K) = \mathrm{Sep}_H(K)$ and $\mathrm{Sim}_{H'}(K) = \mathrm{Sim}_H(K)$ for each leaf clique $K \in \mathcal{L}_{\max}$ in the current reduced chordal graph H'. In other words, not only do the leaves in $\mathcal{L}_{\max}$ 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 the first and last lines inside the **for** loop.

Let $T \setminus \mathcal{L}_T$ be the tree obtained by pruning the set of leaves $\mathcal{L}_T$ from $T \in \mathcal{T}_H$. We let $\mathrm{Sim}_H(\mathcal{L}_T)$ be the union of all simplicial node sets $\mathrm{Sim}_H(K)$ where $K \in \mathcal{L}_T$. The following lemma shows that the algorithm in Figure 5 generates a clique tree. The lemma also plays a key role in our proof that any clique tree generated by the algorithm has minimum diameter.

Lemma 7. *The algorithm in Figure 5 generates a clique tree T for which $\mathcal{L}_T$ is the maximum-cardinality leaf set eliminated by the first major step of the algorithm.*

Proof. It is easy to show that the set $\mathcal{E}_{\min}$ generated by the algorithm is the edge set of a tree T, and we leave it for the reader to verify this. We first show that $\mathcal{L}_T = \mathcal{L}_{\max}$, where $\mathcal{L}_{\max}$ is the maximum-cardinality leaf set eliminated by the first major step of the algorithm. As each clique $K \in \mathcal{L}_{\max}$ is processed in the inner loop, the algorithm adds to $\mathcal{E}_{\min}$ an edge incident on K and on some clique $P \in \mathcal{K}_G \setminus \mathcal{L}_{\max}$. Clearly, the algorithm adds no further edges incident on K *after* its elimination. Since each parent is taken from $\mathcal{K}_G \setminus \mathcal{L}_{\max}$, no step *prior to* that eliminating K adds an edge incident on K. It follows that each clique in $\mathcal{L}_{\max}$ is adjacent to only one clique in

T, and thus $\mathcal{L}_{\max} \subseteq \mathcal{L}_T$. Since $\mathcal{L}_{\max}$ is a maximum-cardinality leaf set, it will follow from the fact that $T \in \mathcal{T}_G$ (proven next) that $\mathcal{L}_T = \mathcal{L}_{\max}$.

The proof that T is indeed a clique tree is by induction on the number of major steps taken by the algorithm. The base step (no iterations through the **while** loop) is trivial. For the induction step, let G be a chordal graph for which the algorithm goes through $k \geq 1$ major steps, and suppose that the algorithm generates a clique tree for any chordal graph requiring fewer than k major steps of the algorithm.

Let $T' = T \backslash \mathcal{L}_{\max}$. Clearly, T' is the tree generated by the subsequent $k-1$ major steps of the algorithm. Moreover, these $k-1$ major steps are precisely the same as applying the algorithm directly to the graph $G \setminus \mathrm{Sim}_G(\mathcal{L}_{\max})$ with no prior elimination step. It follows from the induction hypothesis that T' is a clique tree of the chordal graph $G \setminus \mathrm{Sim}_G(\mathcal{L}_{\max})$. Consequently, Theorem 1 holds in T for every pair of cliques taken from $\mathcal{K}_G - \mathcal{L}_{\max}$. All that remains to be shown is that for every pair of cliques $K \in \mathcal{L}_{\max}$ and $K' \in \mathcal{K}_G - \{K\}$, the intersection $K \cap K'$ is contained in every clique on the path connecting K and K' in T.

Let K and P be, respectively, a leaf clique eliminated during the first major step and the parent clique of K chosen by the algorithm, so that $\{K, P\}$ is an edge added to $\mathcal{E}_{\min}$ during the first major step. Let $K' \in \mathcal{K}_G - \{K\}$. Since $K \in \mathcal{L}_{\max} \subseteq \mathcal{L}_G$ and $\mathrm{Sep}_G(K) \subset P$, we have

$$K \cap K' \subseteq \mathrm{Sep}_G(K) \cap K' \subseteq P \cap K'. \tag{1}$$

If $K' \in \mathcal{K}_G - \mathcal{L}_{\max}$, then by the induction hypothesis $P \cap K'$ is contained in every clique on the path connecting P with K', which proves the result in this case. Suppose however that $K' \in \mathcal{L}_{\max} - \{K\}$, and let P' be the parent clique of K' chosen by the algorithm. Again, since $K' \in \mathcal{L}_{\max} \subseteq \mathcal{L}_G$ and $\mathrm{Sep}_G(K') \subset P'$, we have

$$K' \cap P \subseteq \mathrm{Sep}_G(K') \cap P \subseteq P' \cap P. \tag{2}$$

Combining (1) and (2) we see that $K \cap K' \subseteq P' \cap P$. Moreover, by the induction hypothesis, $P' \cap P$ is contained in every clique on the path connecting P with P', and this, in conjunction with $K \cap K' \subseteq P' \cap P$, suffices to prove that T is a clique tree. $\square$

4.2 Proof of minimum diameter

For a clique tree $T \in \mathcal{T}_G$ and any pair of cliques $K, K' \in \mathcal{K}_G$, let $\mathrm{dist}(K, K')$ be the distance from K to K' along the single path connecting the pair in T. The diameter of T is given by $\mathrm{diam}(T) := \max\{\mathrm{dist}(K, K')\}$, where K and K' range over every distinct pair of leaves taken from $\mathcal{L}_T$.

We are now ready to prove that the algorithm in Figure 5 finds a clique tree $T_{\min}$ that minimizes $\mathrm{diam}(T)$ over all clique trees $T \in \mathcal{T}_G$. To proceed, we show in Lemma 8 that whenever $P = P'$ in the proof of Lemma 2 (see Figure 4), 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 $\mathcal{L}_{\max}$, there exists a minimum-diameter clique tree $T_{\min} \in \mathcal{T}_G$ for which $\mathcal{L}_{T_{\min}} = \mathcal{L}_{\max}$. The main result then follows by a simple induction argument.

LEMMA 8. *Assume* $|\mathcal{K}_G| \geq 3$. *Let* $K \in \mathcal{L}(S)$ *and suppose* K *is not a leaf in some clique tree* $T \in \mathcal{T}_G$. *Let* P *be any neighbor of* K *in* T *such that* $S \subset P$. *There exists then a clique tree* $T' \in \mathcal{T}_G$ *for which the following properties hold:*

(1) K *is a leaf in* 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) $\operatorname{diam}(T') \leq \operatorname{diam}(T)$.

PROOF. First, note that the existence of a neighbor P of K in T for which $S \subset P$ is ensured by Theorem 1. Now consider the restructured clique tree T' produced in the proof of Lemma 2 when $P = P'$ (see Figure 9). That

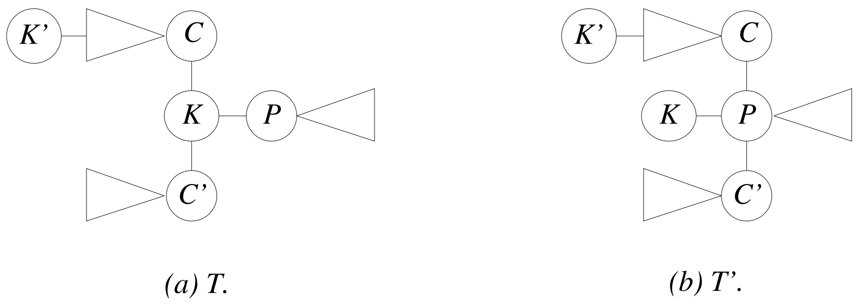


Fig. 9: Transformation of T into T' in which K is a leaf and for which $\operatorname{diam}(T') \leq \operatorname{diam}(T)$, as discussed in the proof of Lemma 8.

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 a node K' in one of the moved subtrees. The path connecting K and K' in the restructured tree is, however, no longer than the path connecting K' and P in the original tree. It follows that making all the neighbors of K (except P) neighbors of P cannot increase the diameter. □

LEMMA 9. *Assume* $|\mathcal{K}_G| \geq 3$, *and let* $T \in \mathcal{T}_G$ *be any clique tree for which* $|\mathcal{L}_T| \geq |\mathcal{L}_{T'}|$ *for all* $T' \in \mathcal{T}_G$. *There exists then a minimum-diameter clique tree* $T_{\min} \in \mathcal{T}_G$ *such that* $\mathcal{L}_{T_{\min}} = \mathcal{L}_T$.

PROOF. Let $T \in \mathcal{T}_G$ be chosen as in the premise. Choose a minimum-diameter clique tree $T_{\min} \in \mathcal{T}_G$ for which $\mathcal{L}_{T_{\min}}$ contains as many of the leaf cliques belonging to $\mathcal{L}_T$ as possible. By way of contradiction, assume that $\mathcal{L}_{T_{\min}} \neq \mathcal{L}_T$. By Lemma 5, since $|\mathcal{L}_T| \geq |\mathcal{L}_{T_{\min}}|$, there exists a leaf clique $K \in \mathcal{L}_T - \mathcal{L}_{T_{\min}}$. Suppose that $K \in \mathcal{L}(S)$. Since $|\mathcal{K}_G| \geq 3$ and $|\mathcal{K}(S)| \geq 2$, at least one clique $K' \in \mathcal{K}(S)$ is *not* a leaf in T. Now consider the subtree of $T_{\min}$ induced by $\mathcal{K}(S)$. Let $P \in \mathcal{K}(S)$ be the clique adjacent to K along the path from K to K' in the subtree of $T_{\min}$ induced by $\mathcal{K}(S)$ (possibly $P = K'$). Observe that if $P = K'$, then P is not a leaf in T, and if $P \neq K'$, then P is not a leaf in $T_{\min}$. It follows that P is not one of the leaf cliques that T and $T_{\min}$ have in common. Thus, using Lemma 8 to restructure $T_{\min}$ 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 contradicts our assumption about $T_{\min}$, thereby proving the result. □

Remark 5. Some clique trees of minimum diameter do not have maximum-cardinality leaf sets; some clique trees that have maximum-cardinality leaf sets are not of minimum diameter. We leave it for the reader to supply examples that confirm these statements.

Theorem 3. *The algorithm in Figure 5 generates a clique tree of minimum diameter.*

Proof. That the algorithm generates a clique tree was proven in Lemma 7. We prove by induction on $m = |\mathcal{K}_G|$ that the clique tree has minimum diameter. 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 7, $\mathcal{L}_{T_{\text{alg}}}$ 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 3 and 4 imply that the first major step eliminates the nodes in $\text{Sim}_G(\mathcal{L}_{T_{\text{alg}}})$. Clearly, $T_{\text{alg}} \setminus \mathcal{L}_{T_{\text{alg}}}$ 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 $G \setminus \text{Sim}_G(\mathcal{L}_{T_{\text{alg}}})$ with no prior elimination step. It follows from the induction hypothesis that $T_{\text{alg}} \setminus \mathcal{L}_{T_{\text{alg}}}$ is a minimum-diameter clique tree of the chordal graph $G \setminus \text{Sim}_G(\mathcal{L}_{T_{\text{alg}}})$.

By Lemma 9, there exists a minimum-diameter clique tree $T_{\min} \in \mathcal{T}_G$ such that $\mathcal{L}_{T_{\min}} = \mathcal{L}_{T_{\text{alg}}} = \mathcal{L}_{\max}$. Thus, $T_{\text{alg}} \setminus \mathcal{L}_{T_{\text{alg}}}$ and $T_{\min} \setminus \mathcal{L}_{T_{\min}}$ are both clique trees of $G \setminus \text{Sim}_G(\mathcal{L}_{T_{\text{alg}}})$. It follows from the induction hypothesis that $\text{diam}(T_{\text{alg}} \setminus \mathcal{L}_{T_{\text{alg}}}) \leq \text{diam}(T_{\min} \setminus \mathcal{L}_{T_{\min}})$. 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 have

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

which proves the result. □

Note that the clique tree constructed by applying the algorithm to the graph in Figure 1 has diameter four (see Figure 8), whereas the clique tree that is a path has diameter six (see Figure 2). Moreover, it follows from Theorem 3 that four is the minimum diameter over all clique trees for this graph.

5. A linear-time implementation

The key step in the algorithm in Figure 5 is the selection of a maximum-cardinality leaf set $\mathcal{L}_{\max} \subseteq \mathcal{L}_H$; the other lines in the main loop merely remove the cliques in $\mathcal{L}_{\max}$ and collect the edges of the minimum-diameter clique tree. Section 5.1 introduces a simple algorithm which combines the selection and elimination of the cliques in $\mathcal{L}_{\max}$ into a single process. In Section 5.2 we use the new algorithm for computing $\mathcal{L}_{\max}$ to obtain a detailed version of the algorithm in Figure 5. Finally, Section 5.3 shows how a linear-time implementation of the detailed algorithm can be achieved.

5.1 Generating a maximum-cardinality leaf set

This section introduces a practical algorithm for generating and removing a maximum-cardinality leaf set $\mathcal{L}_{\max}$. To describe the algorithm we need the following parameters associated with each clique in the graph. For $K \in \mathcal{K}_G$, we define the parameter $\rho(K)$ to be the number of simplicial nodes contained in K, and we define $\sigma(K)$ to be the size of the largest separator in $\overline{\mathcal{S}}(K)$. Lemma 10 gives a useful formula for $\sigma(K)$.

LEMMA 10. *For $K \in \mathcal{K}_G$, we have*

$$\sigma(K) = \max\Big\{|K \cap K'|, \text{ where } K' \in \mathcal{K}_G - \{K\}\Big\}.$$

PROOF. Choose two distinct cliques $K, K' \in \mathcal{K}_G$. Applying Theorem 1 to any clique tree $T \in \mathcal{T}_G$, we have $K \cap K' \subseteq S$ for some separator $S \in \mathcal{S}(K)$; hence, $\sigma(K) \geq |K \cap K'|$. The definition of $\sigma(K)$ implies that there exists $K'' \in \mathcal{K}_G - \{K\}$ such that $\sigma(K) = |K \cap K''|$, and this completes the proof. $\square$

The algorithm to calculate $\mathcal{L}_{\max}$, shown in Figure 10, processes the members of $\mathcal{L}_H$ in some arbitrary order. (Computing $\mathcal{L}_H$ will be discussed in Section 5.2.) To test $K \in \mathcal{L}_H$ for inclusion in $\mathcal{L}_{\max}$, 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 $\mathcal{L}_{\max} \subseteq \mathcal{L}_H$. First, Lemma 11 gives a useful condition that holds if and only if $\sigma_{H'}(K) = \sigma_H(K)$, and then Theorem 4 proves the algorithm in Figure 10 correct.

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

Fig. 10: Algorithm for generating a maximum-cardinality leaf set.

LEMMA 11. *For some leaf separator S, choose $K \in \mathcal{L}_H(S) \subseteq \mathcal{L}_H$. When the algorithm in Figure 10 tests K for inclusion in $\mathcal{L}_{\max}$, we have $\sigma_{H'}(K) = \sigma_H(K)$ if and only if $|\mathcal{K}_{H'}(S)| \geq 2$.*

PROOF. Let $K \in \mathcal{L}_H(S) \subseteq \mathcal{L}_H$, and consider the iteration of the algorithm that processes K. Since $\overline{\mathcal{S}}_H(K) = \{S\}$, clearly $\sigma_H(K) = |S|$.

If $|\mathcal{K}_{H'}(S)| \geq 2$, then by Lemma 10, $\sigma_{H'}(K) \geq |S|$. Since $\mathcal{K}_{H'} \subseteq \mathcal{K}_H$, it follows from Lemma 10 that $\sigma_{H'}(K) \leq \sigma_H(K) = |S|$. Thus, $\sigma_{H'}(K) = \sigma_H(K)$.

If $\sigma_{H'}(K) = \sigma_H(K)$, then $|K \cap K'| = |S|$ for some clique $K' \in \mathcal{K}_{H'} - \{K\}$. From Theorem 1 we have $K \cap K'' \subseteq S$ for every clique $K'' \in \mathcal{K}_H - \{K\}$. Consequently, $K \cap K' = S$, and therefore $K, K' \in \mathcal{K}_{H'}(S)$, which proves the result. □

THEOREM 4. *The algorithm in Figure 10 computes a maximum-cardinality leaf set $\mathcal{L}_{\max}$.*

PROOF. Consider the partition of $\mathcal{L}_H$ into leaf-cohort sets $\mathcal{L}_H(S)$ where S ranges over the set of distinct leaf separators. Lemma 5 gives the two conditions that must be satisfied by $\mathcal{L}_{\max}$: 1) whenever $\mathcal{L}_H(S) \subset \mathcal{K}_H(S)$, every clique in $\mathcal{L}_H(S)$ must be included in $\mathcal{L}_{\max}$, and 2) whenever $\mathcal{L}_H(S) = \mathcal{K}_H(S)$, precisely one clique in $\mathcal{L}_H(S)$ must be excluded from $\mathcal{L}_{\max}$. Consider an arbitrary leaf-cohort set $\mathcal{L}_H(S) = \{K_1, K_2, \ldots, K_t\} \subseteq \mathcal{L}_H$, 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 $|\mathcal{K}_{H'}(S)| \geq 2$ when the algorithm processes K_i, $1 \leq i \leq t-1$. Therefore, by Lemma 11, $\sigma_{H'}(K_i) = \sigma_H(K_i)$, and K_i is included in $\mathcal{L}_{\max}$. We now consider whether or not the algorithm includes K_t in $\mathcal{L}_{\max}$. There are two cases to consider. First, suppose $\mathcal{L}_H(S) = \mathcal{K}_H(S)$. It follows that $\mathcal{K}_{H'}(S) = \{K_t\}$ when the algorithm finally examines K_t. Consequently, by Lemma 11, $\sigma_{H'}(K_t) \neq \sigma_H(K_t)$, and K_t is therefore excluded from $\mathcal{L}_{\max}$, as required.

Now, suppose $\mathcal{L}_H(S) \subset \mathcal{K}_H(S)$ and consider the following two subcases. First, assume $\mathcal{K}_H(S) \nsubseteq \mathcal{L}_H$. In this case, $|\mathcal{K}_{H'}(S)| \geq 2$ when the algorithm examines K_t, and by Lemma 11, K_t is included in $\mathcal{L}_{\max}$, as required. Now, assume $\mathcal{K}_H(S) \subseteq \mathcal{L}_H$. Let $K' \in \mathcal{L}_H$ be chosen so that $K' \in \mathcal{K}_H(S) - \mathcal{L}_H(S)$. It follows that $K' \in \mathcal{L}_H(S')$ where $S \subset S'$. The key idea is to choose K' that maximizes $|S'|$ among all the leaf separators S' for which $S \subset S'$. Since $S \subset S'$, we have $\mathcal{K}_H(S') \subseteq \mathcal{K}_H(S) \subseteq \mathcal{L}_H$. It suffices to show that $\mathcal{K}_H(S') = \mathcal{L}_H(S')$, for we have shown in the previous paragraph that if this were true, the last member of $\mathcal{L}_H(S')$ to be processed by the algorithm would be excluded from $\mathcal{L}_{\max}$, and thus retained in the graph H' when K_t is examined. Consequently, we would have $|\mathcal{K}_{H'}(S)| \geq 2$, and thus K_t would be included in $\mathcal{L}_{\max}$, as required. To verify that $\mathcal{L}_H(S') = \mathcal{K}_H(S')$, assume that $K'' \in \mathcal{K}_H(S') - \mathcal{L}_H(S')$. Since $\mathcal{K}_H(S') \subseteq \mathcal{L}_H$, 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 Detailed algorithm

In this subsection we incorporate the algorithm for removing a maximum-cardinality leaf set, shown in Figure 10, into the high-level minimum-diameter clique tree algorithm shown in Figure 5. Figure 11 details an algorithm based on this approach. Proceeding through the algorithm from beginning to end, we discuss the key implementation issues connected with this approach.

5.2.1 Implementing the maximum-cardinality leaf set algorithm

The initialization loop uses the parameters introduced in the previous subsection to compute $\mathcal{L}_H$. The following result ensures that $\mathcal{L}_H$ is computed correctly.

LEMMA 12. *$K \in \mathcal{L}_H$ if and only if $\sigma_H(K) + \rho_H(K) = |K|$.*

PROOF. Suppose $K \in \mathcal{L}_H$ and let S be the single separator in $\overline{\mathcal{S}}_H(K)$. From Theorem 1 and the fact that $\overline{\mathcal{S}}_H(K) = \{S\}$, we have $K \cap K' \subseteq S$ for every clique $K' \in \mathcal{K}_H - \{K\}$. Hence, any node $v \in K - S$ belongs to no other clique in the graph. Thus, we have $\rho_H(K) \geq |K - S|$, and since $|\mathcal{K}_H(S)| \geq 2$, none of the nodes in S is simplicial. Hence $\rho_H(K) = |K - S|$, which along with $\sigma_H(K) = |S|$, proves necessity.

To prove sufficiency, suppose $\sigma_H(K) + \rho_H(K) = |K|$. Choose $S \in \overline{\mathcal{S}}_H(K)$ such that $|S| = \sigma_H(K)$. Since $\rho_H(K) = |K| - \sigma_H(K)$ 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 $\mathcal{M}_H$. Consequently, $\overline{\mathcal{S}}_H(K) = \{S\}$, and by Lemma 3, $K \in \mathcal{L}_H$. □

Because the high-level algorithm in Figure 5 selects $\mathcal{L}_{\max}$ before any clique in $\mathcal{L}_{\max}$ is eliminated, it easily identifies a set of clique-tree edges to be added by the major step. The detailed algorithm in Figure 11 differs from

```
        [Initialization]
01      $\mathcal{E}_{\min} \leftarrow \emptyset$ ;
02      $H' \leftarrow H \leftarrow G$ ; $\mathcal{L}_H \leftarrow \emptyset$ ;
03      for $K \in \mathcal{K}_H$ do
04          $\sigma_H(K) \leftarrow \sigma_{H'}(K) \leftarrow \sigma_G(K)$ ; $\rho_H(K) \leftarrow \rho_{H'}(K) \leftarrow \rho_G(K)$ ;
05          if $\sigma_H(K) + \rho_H(K) = |K|$ then $\mathcal{L}_H \leftarrow \mathcal{L}_H \cup \{K\}$ ;
06      end for ;
07      Set up empty parent-child data structure ;

08      while $|\mathcal{K}_H| \geq 3$ do

            [Compute and eliminate $\mathcal{L}_{\max}$]
09          for $K \in \mathcal{L}_H$ do
10              if $\sigma_{H'}(K) = \sigma_H(K)$ do
11                  Choose $P \in \mathcal{K}_{H'} - \{K\}$ for which $\text{Sep}_H(K) \subset P$ ;
12                  Use $K$ and $P$ to update parent-child data structure ;
13                  $H' \leftarrow H' \setminus \text{Sim}_H(K)$ ;
14                  Compute new values for $\sigma_{H'}(P)$ and $\rho_{H'}(P)$ ;
15              end if ;
16          end for ;

            [Prepare for next iteration of while]
17          $H \leftarrow H'$ ; $\mathcal{L}_H \leftarrow \emptyset$ ;
18          for each parent clique $P$ in the parent-child data structure
19              $\sigma_H(P) \leftarrow \sigma_{H'}(P)$ ; $\rho_H(K) \leftarrow \rho_{H'}(K)$ ;
20              if $\sigma_H(P) + \rho_H(P) = |P|$ then $\mathcal{L}_H \leftarrow \mathcal{L}_H \cup \{P\}$ ;
21          end for ;
22          Extract edges from parent-child data structure and add to $\mathcal{E}_{\min}$ ;
23          Set up empty parent-child data structure ;

24      end while ;

25      if $|\mathcal{K}_H| = 2$ do
26          $\mathcal{E}_{\min} \leftarrow \mathcal{E}_{\min} \cup \{K, P\}$ where $\{K, P\} = \mathcal{K}_H$ ;
27      end if ;
```

Fig. 11: Detailed algorithm for generating a minimum-diameter clique tree.

the high-level algorithm in this regard; it uses the algorithm in Figure 10 to both compute and eliminate a maximum-cardinality leaf set. (See the first **for** loop inside the **while** loop in Figure 11.) With $\mathcal{L}_{\max}$ not known in advance, the detailed algorithm can add edges to $\mathcal{E}_{\min}$ only after this **for** loop has completed the computation of $\mathcal{L}_{\max}$. This is achieved by maintaining a *parent-child data structure* that stores the cliques in $\mathcal{L}_{\max}$ and the required edges *upon completion of the* **for** *loop.* In line 12, the leaf clique K and its "candidate" parent P (chosen in line 11) are incorporated into the parent-child data structure. The edges are extracted from the data structure and added to $\mathcal{E}_{\min}$ in line 22.

5.2.2 The parent-child data structure

The parent-child data structure comprises a set $\mathcal{P}$ of current "parent" cliques, along with a nonempty set $\mathcal{C}(P)$ of current "children" for each parent $P \in \mathcal{P}$. Upon entry into the loop, all of these sets are empty (lines 7 and 23). For each leaf clique K that passes the test for membership in $\mathcal{L}_{\max}$, a candidate parent clique $P \in \mathcal{K}_{H'} \setminus \{K\}$ for which $\mathrm{Sep}_H(K) \subset P$ is found. (How to locate this candidate parent clique in constant time will be discussed in Section 5.3.1.) Given K and P, the following lines of code implement the parent-child data structure update in line 12.

```
$\mathcal{P} \leftarrow \mathcal{P} \cup \{P\}$ ;
**if** $K \in \mathcal{P}$ **then** $\mathcal{P} \leftarrow \mathcal{P} - \{K\}$ ;
$\mathcal{C}(P) \leftarrow \mathcal{C}(P) \cup \{K\} \cup \mathcal{C}(K)$ ;
$\mathcal{C}(K) \leftarrow \emptyset$ ;
```

To prove that the data structure ultimately stores the edges needed by $\mathcal{E}_{\min}$, it suffices to show that three properties hold *upon completion of the data structure* (i.e., upon completion of the **for** loop). The first two are trivial; we leave it for the reader to confirm that

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

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

The following lemma provides the third required property.

LEMMA 13. *Upon completion of the parent-child data structure, we have* $\mathrm{Sep}_H(K) \subset P$ *for every clique* $K \in \mathcal{C}(P)$*, where* $P \in \mathcal{P}$.

PROOF. The following simple induction argument suffices. The result holds vacuously before the first iteration of the loop is begun. We now 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 clique in $\mathcal{K}_{H'} \setminus \{K\}$ for which $\mathrm{Sep}_H(K) \subset P$. 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. The algorithm explicitly chose P so that the property holds for K (line 11). Now, choose $C \in \mathcal{C}(K)$. Note that $C \in \mathcal{L}_H$. By induction we have $\mathrm{Sep}_H(C) \subseteq K$. Moreover, since $K \in \mathcal{L}_H$, we have $\mathrm{Sep}_H(C) \subseteq \mathrm{Sep}_H(K)$, which, in turn, is properly contained in P. Consequently, $\mathrm{Sep}_H(C) \subset P$, 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 Lemma 13 and the discussion preceding it that the edges represented by the sets $\mathcal{P}$ and $\mathcal{C}(P)$, where $P \in \mathcal{P}$, are precisely the edges that should be added to $\mathcal{E}_{\min}$. These edges moreover are added to $\mathcal{E}_{\min}$ in line 22.

5.2.3 Preparation for the next major step

Next we show why $\sigma_{H'}(P)$ and $\rho_{H'}(P)$ in line 14 are the only parameters that might require updating after a graph reduction step in line 13. More specifically, we show that after a graph reduction step, the only clique $K' \in \mathcal{K}_{H'}$ for which the values of $\sigma_{H'}(K')$ or $\rho_{H'}(K')$ may have changed is the candidate parent P.

LEMMA 14. *Let $K \in \mathcal{L}_H(S)$ and $H' = H \setminus \mathrm{Sim}_H(K)$. If $\sigma_{H'}(P) \neq \sigma_H(P)$ or $\rho_{H'}(P) \neq \rho_H(P)$ for a clique $P \in \mathcal{K}_{H'}$, then $\mathcal{K}_H(S) = \{K, P\}$.*

PROOF. Assume $\sigma_{H'}(P) \neq \sigma_H(P)$. Since $\mathcal{K}_{H'} = \mathcal{K}_H \setminus \{K\}$, it follows from Lemma 10 that $\sigma_{H'}(P) \leq \sigma_H(P)$; whence $\sigma_{H'}(P) < \sigma_H(P)$. It follows that any separator $S' \in \overline{\mathcal{S}}_H(P)$ for which $\sigma_H(P) = |S'|$ is not a member of $\mathcal{M}_{H'}$. From Property 1 of Lemma 6, the multiset of separators of the reduced graph $H' = H \setminus \mathrm{Sim}_H(K)$ is given by $\mathcal{M}_H - \{S\}$. It follows that S' is unique and moreover $S' = S$; thus $\sigma_H(P) = |S|$ and $\{K, P\} \subseteq \mathcal{K}_H(S)$. Now, were it the case that $|\mathcal{K}_{H'}(S)| \geq 2$, we would have $\sigma_{H'}(P) \geq |S| = \sigma_H(P)$, which is impossible since $\sigma_{H'}(P) < \sigma_H(P)$.

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

The second **for** loop inside the **while** loop initializes data for the next pass through the **while** loop. From Lemma 14 we know that the only cliques for which $\sigma_{H'}(K) \neq \sigma_H(K)$ or $\rho_{H'}(K) \neq \rho_H(K)$ upon entry into the loop are the cliques belonging to $\mathcal{P}$. It is therefore sufficient to record for each clique $K \in \mathcal{P}$ new values $\sigma_H(K)$ and $\rho_H(K)$ to be used in the next major step. In the next result we show that the leaf cliques $\mathcal{L}_H$ are also found among the parent cliques recorded in $\mathcal{P}$ (line 20).

LEMMA 15. *Upon entering the* **for** *loop that prepares for the next major step of the algorithm in Figure 11, we have $\mathcal{L}_{H'} \subseteq \mathcal{P}$.*

PROOF. Assume the algorithm is entering the loop, and let $K \in \mathcal{L}_{H'}$. If $K \in \mathcal{L}_{H'} - \mathcal{L}_H$, then it follows from Lemma 12 that $\sigma_{H'}(K) \neq \sigma_H(K)$ or $\rho_{H'}(K) \neq \rho_H(K)$, and by Lemma 14 we have $K \in \mathcal{P}$. Now assume $K \in \mathcal{L}_{H'} \cap \mathcal{L}_H$. It follows that the algorithm excluded K from $\mathcal{L}_{\max}$, because $\sigma_{H'}(K) \neq \sigma_H(K)$ in line 10 when K was considered for inclusion in $\mathcal{L}_{\max}$. So again, by Lemma 14, we have $K \in \mathcal{P}$. □

5.3 Complexity

To facilitate our discussion of the algorithm's time complexity, let $n := |V|$, $e := |E|$, $m := |\mathcal{K}_G|$, and $q := \sum_{i=1}^{m} |K_i|$. It is well known that $m \leq n$ and $q \leq e$ [11]. In this section we will show that the algorithm in Figure 11 can be implemented to run in $O(n+e)$ time. Below we analyze the time complexity for most of the operations performed by the algorithm, postponing two less obvious complexity issues to be addressed in two subsections which conclude this one.

Clearly, the algorithm requires as input the set of cliques $\mathcal{K}_G$ and the parameters $\sigma_G(K)$ and $\rho_G(K)$, for $K \in \mathcal{K}_G$. A clique tree $T = (\mathcal{K}_G, \mathcal{E}) \in \mathcal{T}_G$ 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 [3, 22, 25]. It is straightforward to compute the parameters $\sigma_G(K)$ and $\rho_G(K)$ from T in $O(m + q)$ time, and thus the required input can be obtained in $O(n + e)$ time.

Next, we verify that the body of each **for** loop is executed a total of $O(m)$ times. The body of the initialization loop clearly is executed m times. It follows directly from Lemma 5 that for each major step we have $|\mathcal{L}_H| \leq 2|\mathcal{L}_{\max}|$; hence, the body of the loop that computes and eliminates $\mathcal{L}_{\max}$ is executed no more than $2m$ times. Finally, since $|\mathcal{P}| \leq |\mathcal{L}_{\max}|$, the body of the loop that prepares for the next major step is executed no more than m times.

We now determine the total cost associated with individual operations within the **for** loops. Each assignment statement involving the σ's and the ρ's is clearly a constant time operation. Similarly, the cost of evaluating each **if** expression involving the σ's and the ρ's is constant. Implementing the leaf sets $\mathcal{L}_H$ as singly-linked lists restricts each operation associated with these sets to constant time. Thus, the total time required to execute the first and last **for** loops is $O(m)$.

The parent-child data structure can be implemented so that the total cost of all operations associated with it is $O(m)$: the set $\mathcal{P}$ should be implemented as a doubly-linked list to enable insertion and deletion of members in constant time; the sets $\mathcal{C}(P)$, where $P \in \mathcal{P}$, can be implemented as singly-linked lists, with a pointer to the tail of each list to enable performance of the "child merging" in constant time.

With the exception of lines 11 and 14, the above arguments suffice to show that the algorithm in Figure 11 can be implemented in $O(n + e)$ time. Subsection 5.3.1 below discusses use of a particular clique tree representation of the reduced graph to allow efficient computation of candidate parent cliques in line 11. Subsection 5.3.2 describes how a variant of the $\sigma_{H'}$ parameters can be substituted for the original $\sigma_{H'}$ parameters throughout the algorithm in order to allow efficient implementation of line 14.

5.3.1 Using a clique-tree representation of each reduced graph

The algorithm maintains $\mathrm{Sim}_H(K)$ for each clique $K \in \mathcal{K}_H$ by keeping count of the number of cliques to which each node belongs. Initially, those members of $K \in \mathcal{K}_H$ that belong to no other clique are placed in $\mathrm{Sim}_H(K)$. For each leaf clique $K \in \mathcal{L}_H$ those members of K that are not in $\mathrm{Sim}_H(K)$ implicitly form the set $\mathrm{Sep}_H(K)$. Whenever the algorithm performs the graph elimination step $H' \leftarrow H' \setminus \mathrm{Sim}_H(K)$, it decrements the "clique count" for each node in $\mathrm{Sep}_H(K)$, and places any new simplicial nodes in $\mathrm{Sim}_{H'}(P)$, where P was chosen in line 11. This can be done using a clique tree to represent G and each subsequent reduced graph H' [19, 25]. Using techniques described in these papers the total work required for these tasks is $O(n+q)$.

Now consider the task of determining the candidate parent P for which $\mathrm{Sep}_H(K) \subset P$. Maintaining the clique-tree representation $T' \in \mathcal{T}_{H'}$ enables efficient implementation of this step. It follows from Lemma 1 that a candidate parent P can be found among the neighbors of K in T'. Note that restricting the search to all neighbors of K in T' is not sufficient to obtain a linear-time algorithm: we need access to P in constant time. We now briefly discuss a data structure used to represent clique trees in Lewis et al. [19], which provides this capability.

The initial data structure is a *rooted* clique tree $T \in \mathcal{T}_G$ with the nodes in each clique listed in ascending order by some perfect elimination ordering of the underlying chordal graph. (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 in the cliques can be done in $O(q)$ total time, and sorting the children in their lists can be done in $O(n)$ total time, both by careful application of a bucket sort.

Let $T' \in \mathcal{T}_{H'}$ be a clique-tree representation of H' obtained using the techniques described in Lewis et al. [19]. Due to the initial ordering of the children and the rules governing the "child-list" updating, we have the following: For each $K \in \mathcal{K}_{H'} - \mathcal{L}_{T'}$ and its "first child" C_1, the separator $K \cap C_1$ is maximal among the separators in K; that is,

$$C_1 \cap K \in \overline{\mathcal{S}}_{H'}(K), \qquad \text{where } C_1 \text{ is the first child of } K. \tag{3}$$

Let $K \in \mathcal{L}_{H'}(S)$, where $S = \mathrm{Sep}_{H'}(K)$. If K has no children in T', then K has a parent K' in T', which is the only clique adjacent to K in T'. It follows from Lemma 1 that $\mathrm{Sep}_{H'}(K) \subset K'$; hence K' may serve as the candidate parent in line 11 of the algorithm. If, on the other hand, K has a child in T', it then follows from Equation (3) and the fact that $|\overline{\mathcal{S}}_{H'}(K)| = 1$ that the first child C_1 may serve as the candidate parent P. We can therefore locate P in constant time.

5.3.2 Using a variant of $\sigma_{H'}$

As before, it follows from Lemma 1 that $\sigma_{H'}(P)$ can be updated by examining only the neighbors of K in T'. Since P may not be a leaf clique, it may have several maximal separators, and hence the maximal separator shared with its first child C_1 may not be maximum. In fact, we know of no way to compute $\sigma_{H'}(K)$ in constant time, even using the clique-tree representation described in the previous subsection. We can nonetheless work around this problem by replacing the σ parameters throughout the algorithm with slightly different parameters, which are adequate for determining membership in $\mathcal{L}_H$ and $\mathcal{L}_{\max}$, but can be computed more efficiently than the σ parameters.

For any clique $K \in \mathcal{K}_{H'}$, let $\widehat{\sigma}_{H'}(K) = |S|$, where S is chosen *arbitrarily* from $\overline{\mathcal{S}}_{H'}(K)$. Consider the algorithm obtained by replacing all the σ's with $\widehat{\sigma}$'s in Figure 11. The *new test* for inclusion in $\mathcal{L}_H$ is $\widehat{\sigma}_H(K) + \rho_H(K) = |K|$. It follows from Lemma 3 that

$$\widehat{\sigma}_H(K) = \sigma_H(K) \qquad \text{for } K \in \mathcal{L}_H. \tag{4}$$

Consequently, every leaf clique is included in the set computed using the new test. For any clique K that fails the *original test* for inclusion in $\mathcal{L}_H$, we have $\sigma_H(K) < |K| - \rho_H(K)$. Since by definition

$$\widehat{\sigma}_H(K) \leq \sigma_H(K), \tag{5}$$

it follows that the new test also excludes K from $\mathcal{L}_H$. In consequence, the new test and the original test are equivalent.

Now consider the new test for inclusion in $\mathcal{L}_{\max}$ [i.e., $\widehat{\sigma}_{H'}(K) = \widehat{\sigma}_H(K)$]. Note that this test is performed only on cliques $K \in \mathcal{L}_H$, and for any such clique we have that $\widehat{\sigma}_H(K) = \sigma_H(K)$. It follows then that that the new test and the original test [i.e., $\sigma_{H'}(K) = \sigma_H(K)$] are equivalent. Having shown that each of the two new tests is equivalent to the corresponding original test, we have shown that the algorithm with the $\widehat{\sigma}$ parameters is equivalent to the one with the σ parameters.

The reason for introducing the $\widehat{\sigma}$ parameters into the algorithm is that they make it possible to implement it to run in linear time. Recall that the first child C_1 (in T') of any clique $K \in \mathcal{K}_{H'}$ is joined to K by a separator in $\overline{\mathcal{S}}_{H'}(K)$ [see Equation (3)]. As a result, $\widehat{\sigma}_H(P)$ can be set to the size of the separator shared with C_1, whenever P has one or more children in T'; otherwise, it can be set to the size of the separator shared with its parent in T'. Consequently, $\widehat{\sigma}_{H'}(P)$, where P is the candidate parent chosen in line 14 of the algorithm, can be updated in constant time.

From the arguments given in this section, it follows that the modified 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 [19, 21]. 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 on 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 in Figure 11 would maintain a clique-tree representation of the *current* chordal graph. (This clique tree may or may not be of minimum diameter.) Lewis et al. [19] 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 in 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 [14] 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, 20]. 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 [18] 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. [8, 9] describe aspects of the Pr component of M that render certain sections of the data propagation computationally independent. This data independence can be exploited 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 would be advantageous 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.

Acknowledgements

We would like to thank Eduardo D'Azevedo, John Gilbert, Eric Kirsch, Esmond Ng, and an anonymous referee for many valuable comments and suggestions. We also would like to thank Alex Pothen for pointing out

an oversight in an earlier version of this paper, which we address here in Section 5.3.2.

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 and B. W. Peyton. An introduction to chordal graphs and clique trees. In J. A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computations*, pages 1–30. Springer Verlag, 1993. IMA Volumes in Mathematics and its Applications, Vol. 56.

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

[9] 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.

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

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

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

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

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

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

[16] 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.

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

[18] 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.

[19] 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.

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

[21] 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.

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

[23] 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.

[24] Y. Shibata. On the tree representation of chordal graphs. *J. Graph Theory*, 12:421–428, 1988.

[25] 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.

[26] J.R. Walter. Representations of chordal graphs as subtrees of a tree. *J. Graph Theory*, 2:265–267, 1978.

Appendix A. Notation

The following table describes the key items in our notation.

G	–	A chordal graph; $G = (V, E)$.
K	–	A maximal clique in G.
$\mathcal{K}_G$	–	The maximal cliques in G.
$\mathcal{K}(S)$	–	The maximal cliques containing $S \subseteq V$.
T	–	A clique tree of G; $T = (\mathcal{K}_G, \mathcal{E})$.
$\mathcal{T}_G$	–	The clique trees of G.
S	–	A minimal node-pair separator of G.
$\mathcal{M}_G$	–	The *multiset* of such separators of G.
$\mathcal{S}(K)$	–	The *set* of such separators included in clique K.
$\overline{\mathcal{S}}(K)$	–	The *set* of such separators maximal among those included in K.
$\mathcal{L}_T$	–	The leaves in a clique tree T.
$\mathcal{L}_G$	–	The leaf cliques in a chordal graph G.
$\mathcal{L}_{\max}$	–	A maximum-cardinality leaf set.
$\mathcal{L}(S)$	–	The leaf cliques $K \in \mathcal{L}_G$ for which $\overline{\mathcal{S}}(K) = \{S\}$.
$\sigma(K)$	–	The size of the largest separator in $\overline{\mathcal{S}}(K)$.
$\rho(K)$	–	The number of simplicial nodes in clique K.