BIT 34 (1994), 000–000.

# FINDING MINIMUM HEIGHT ELIMINATION TREES FOR INTERVAL GRAPHS IN POLYNOMIAL TIME

BENGT ASPVALL AND PINAR HEGGERNES *†

*Department of Informatics, University of Bergen*
*N–5020, Norway*
*email: bengt@ii.uib.no, pinar@ii.uib.no*

**Abstract.**

The elimination tree plays an important role in many aspects of sparse matrix factorization. The height of the elimination tree presents a rough, but usually effective, measure of the time needed to perform parallel elimination. Finding orderings that produce low elimination trees is therefore important. As the problem of finding minimum height elimination tree orderings is NP-hard, it is interesting to concentrate on limited classes of graphs and find minimum height elimination trees for these efficiently. In this paper, we use clique trees to find an efficient algorithm for interval graphs which make an important subclass of chordal graphs. We first illustrate this method through an algorithm that finds minimum height elimination trees for chordal graphs. This algorithm, although of exponential time complexity, is conceptionally simple and leads to a polynomial-time algorithm for finding minimum height elimination trees for interval graphs.

*AMS subject classifications:* 05C50, 65F50, 65Y05.

*Key words:* Sparse matrix computation, elimination trees, parallel computing, interval graphs.

## 1 Introduction

Many problems that frequently arise in science and engineering, require the solution of large, sparse systems of linear equations. These systems are often in the form $Ax = b$, where $A$ is a symmetric, positive definite matrix. Solving such a system via *Cholesky factorization* involves computing the Cholesky factor $L$, such that $A = LL^T$, and then solving the triangular systems $Ly = b$ and $L^T x = y$. If $A$ is sparse then it is often useful to do some preprocessing on its associated graph $G(A)$. Various parameters of this graph, like the elimination tree height, determine, for example, how efficiently the system can be solved in parallel.

As parallel computers become more and more available, they are being widely used for solving sparse linear systems. To get the most benefit from using parallel computers,

*This work was supported by grants from the Norwegian Research Council.
†Received May 1993. Revised May 1994.

the system to be solved must allow a high level of parallelism. The height of the elimination tree indicates the amount of work that can be done in parallel during Cholesky factorization. A low elimination tree means that more work can be done in parallel, reducing the total amount of time needed.

The problem of minimizing the elimination tree height for general graphs is NP-hard. Apart from trees, little has been known about classes of graphs for which the problem can be solved in polynomial time. Several heuristics for reducing the elimination tree height for general graphs have been proposed. In this paper, we use the idea behind one of these heuristics, nested dissection, to develop a polynomial-time algorithm that finds minimum height elimination trees for interval graphs. Interval graphs are useful for modeling real world problems, and they find application in several research areas.

This paper is organized as follows. The necessary background is introduced in Section 2. In Section 3, we develop a simple and illustrative, but exponential-time algorithm for finding minimum height elimination trees for chordal graphs. This algorithm is modified in Section 4 by using dynamic programming to give a polynomial-time algorithm for a rather restricted subclass of chordal graphs, which we call $\Omega$. In Section 5, a polynomial-time algorithm is developed for interval graphs as a natural extension of $\Omega$. Finally, some concluding remarks are given in Section 6.

## 2 Definitions

The reader is assumed to be familiar with standard graph notation. In this section, we introduce the necessary definitions concerning the elimination process, elimination trees, chordal graphs and clique trees.

### 2.1 Symbolic factorization and elimination trees

Let $A$ be an $n \times n$, symmetric, positive definite matrix. The *associated graph* of $A$ is a graph $G(A) = (V, E)$, where $V = \{v_1, ..., v_n\}$, and $E = \{(v_i, v_j) \mid a_{ij} \neq 0\}$. We will be working on $G(A)$ instead of $A$ throughout this paper. The graph of the Cholesky structure of $A$, $G(L + L^T)$ is called the *filled graph* of $A$ and is denoted by $G^*(A)$.

The filled graph $G^* = (V, E^*)$ is found from $G$ by the following process, which is called the *elimination game:* Repeat the following three steps until there are no more vertices in $G$. 1) Pick a vertex $v$ with least subscript in $G$. 2) Add edges between the neighbors of $v$ that are not already adjacent. 3) Delete $v$ from $G$. These added edges are called the *fill-in* edges. The filled graph $G^*$ is obtained by adding to $G$ all the fill-in edges that occur during the elimination. The fill-in edges correspond to the non-zero elements $l_{ij}$ of $L$, where $a_{ij} = 0$.

The *elimination tree* $T$ of $G$ is defined as follows. Vertex $v_j$ is the parent of vertex $v_i$ in $T$ if $j = \min\{k > i \mid (v_i, v_k) \in E^*\}$. The elimination tree provides information on column dependencies in the Cholesky factor. Low elimination trees imply less dependency and thus more parallelism. Liu gives a thorough examination of elimination trees in [17].

The order in which the vertices of $G$ are eliminated, i. e., the order in which the vertices of $G$ are numbered, is called an *elimination ordering* on $G$ and corresponds to a symmetric permutation of $A$. Different elimination orderings yield different filled graphs and elimination trees. A *perfect elimination ordering* is an ordering that results in no fill-in edges. Finding orderings that minimize the fill-in size and finding orderings that minimize the elimination tree height are NP-hard problems, as shown by Yannakakis in [26] and Pothen in [20].

In this paper, we assume that the graphs we are working with are connected. The results can easily be extended to include disconnected graphs. A *separator* $S \subset V$ is a set of vertices whose removal from $G$ disconnects $G$ into two or more connected components. A separator $S$ is *minimal* if no proper subset of $S$ is a separator in $G$. A *$v$-$w$ separator* $S \subset V$ is a set of vertices whose removal from $G$ cuts every path between $v$ and $w$ in $G$, where $v, w \in V$ and $(v, w) \notin E$. A $v$-$w$ separator $S$ is *minimal* if no proper subset of $S$ is a $v$-$w$ separator. Every minimal separator in $G$ is a minimal $v$-$w$ separator for some $v$ and $w \in V$.

Several heuristics aim to find orderings that result in low elimination trees. One such heuristic that we will be using in this paper is *nested dissection*, which does the following: pick a separator $S$ in $G$, number the vertices in $S$ arbitrarily from $n - |S| + 1$ to $n$, for each component in $G - S$ repeat this process recursively until the components cannot be separated further. By choosing the separators carefully, the height of the elimination tree may be reduced as Manne has shown by the following theorem in [18].

THEOREM 2.1. *For every graph $G$, there exists a nested dissection ordering with minimal separators that produces an elimination tree of minimum height.*

By a nested dissection ordering with minimal separators, we mean that the separators chosen at each step of the algorithm are minimal. No assumptions are made on the size of the resulting components. This theorem provides a basis for the algorithms and results that will be presented later in the paper.

## 2.2 Chordal graphs and clique trees

A *clique* in $G$ is a complete subgraph of $G$. A clique $C$ is *maximal* if no other clique in $G$ contains $C$. Since cliques are complete subgraphs, a clique can be identified solely by the vertices in it. In this paper, we treat cliques as sets of vertices. Thus a clique $C$ in a graph $G$ is a set of vertices that induce a complete subgraph of $G$.

An edge in $G$ is called a *chord* of a cycle if it joins two nonconsecutive vertices on the cycle. A graph is *chordal* if every cycle of length at least four has a chord. Fulkerson and Gross show in [9] that chordal graphs are exactly those graphs with perfect elimination orderings. Thus the chordality of a graph $G$ can be established by finding a perfect elimination ordering for $G$. Tarjan and Yannakakis present a simple linear algorithm for this in [25], which they call *maximum cardinality search.* All filled graphs are chordal. A *simplicial vertex* in a chordal graph $G$, is a vertex whose neighbors induce a clique in $G$. Eliminating a simplicial vertex in $G$ does not cause any fill-in edges. Dirac shows in [8] that a chordal graph is either a complete graph, or has at least two non-adjacent simplicial vertices. Induced subgraphs of chordal graphs are also chordal. The following lemmas show some more results on chordal graphs. Lemma 2.2 is from [8], and Lemma 2.3 is from [23].

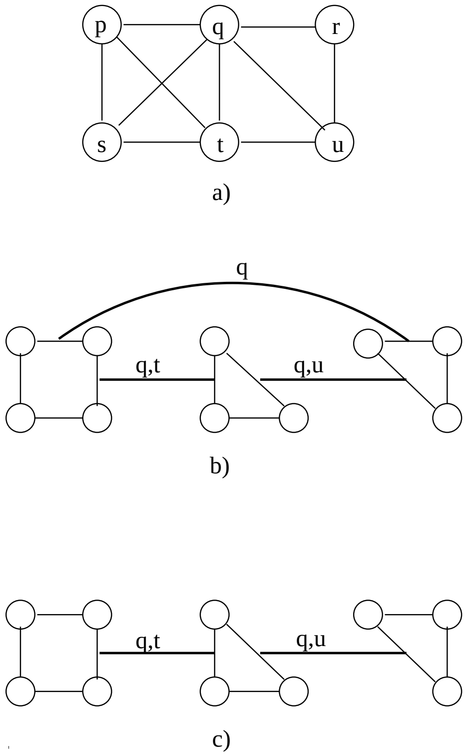


Figure 2.1: a) A chordal graph $G$. b) The clique graph of $G$. c) The clique tree of $G$, which in this case is unique.

Lemma 2.2. *$G$ is a chordal graph if and only if every minimal v-w separator in $G$ is a clique.*

Lemma 2.3. *For every clique $C$ in a chordal graph, there exists a perfect elimination ordering that eliminates the vertices in $C$ last.*

Let $G$ be a chordal graph and let $\mathcal{V}_G = \{C_1, C_2, ..., C_m\}$ be the set of maximal cliques in $G$. The following theorem from [10] will help us define a clique tree for $G$.

Theorem 2.4. *A graph $G$ is chordal if and only if there exists a tree $\mathcal{T} = (\mathcal{V}_G, \mathcal{E})$ that satisfies the following property: for every vertex $v$ in $G$, the set of maximal cliques containing $v$ induces a subtree of $\mathcal{T}$.*

Any such tree $\mathcal{T}$ is called a *clique tree* of $G$. The *clique graph* of $G$ is a graph $\mathcal{G} = (\mathcal{V}_G, \mathcal{E})$, where $\mathcal{E} = \{(C_i, C_j) \mid C_i \cap C_j \neq \emptyset,\ C_i, C_j \in \mathcal{V}_G\}$. Thus each edge $(C_i, C_j)$ in $\mathcal{E}$ corresponds to a nonempty set of vertices $C_i \cap C_j$. The weight of an edge $(C_i, C_j)$ is $|(C_i, C_j)|$. An example is shown in Figure 2.1. A clique tree of $G$ can also be identified by the following theorems from [1] and [3].

Theorem 2.5. *A tree $\mathcal{T} = (\mathcal{V}_G, \mathcal{E})$ is a clique tree of $G$ if and only if $\mathcal{T}$ is a maximum-weight spanning tree of the clique graph of $G$.*

Theorem 2.6. *A tree $\mathcal{T} = (\mathcal{V}_G, \mathcal{E})$ is a clique tree of $G$ if and only if for every pair of* *distinct cliques $C_i$ and $C_j \in \mathcal{V}_G$, the intersection $C_i \cap C_j$ is contained in every maximal clique on the path connecting $C_i$ and $C_j$ in $\mathcal{T}$.*

Corollary 2.7 follows directly from Theorem 2.6.

COROLLARY 2.7. *A tree $\mathcal{T} = (\mathcal{V}_G, \mathcal{E})$ is a clique tree of $G$ if and only if for every pair of distinct cliques $C_i$ and $C_j \in \mathcal{V}_G$, the intersection $C_i \cap C_j$ is contained in every edge on the path connecting $C_i$ and $C_j$ in $\mathcal{T}$.*

As can be seen from the previous definitions and theorems, the clique graph of $G$ is unique whereas Ho and Lee show in [16] that $G$ may have exponentially many clique trees. It is shown in [8] that the number of maximal cliques in a chordal graph $G$ is less than or equal to the number of vertices in $G$. Thus $|\mathcal{V}_G| \leq |V|$.

There is a close relationship between clique trees and separators of $G$, which can be seen in the following results. Lemma 2.8 and Corollary 2.9 are from [16], and are quoted without proofs.

LEMMA 2.8. *Let $G$ be a chordal graph and let $\mathcal{T}$ be a clique tree of $G$. Every edge $(C_i, C_j)$ in $\mathcal{T}$ is a minimal $v$-$w$ separator in $G$, and for every minimal $v$-$w$ separator $S$ in $G$ there is at least one edge $(C_i, C_j) = S$ in $\mathcal{T}$.*

COROLLARY 2.9. *If a chordal graph $G$ has $m$ distinct maximal cliques, then the number of minimal $v$-$w$ separators in $G$ is less than or equal to $m - 1$.*

Let $G$ be a chordal graph with $n$ vertices and $m$ maximal cliques. Since $m \leq n$, by Corollary 2.9, the number of minimal $v$-$w$ separators in $G$ is less than or equal to $n-1$. Recall that both the edges and the vertices in a clique tree of $G$ are cliques in $G$.

In the next section, we will use the results presented here to find minimum height elimination trees for chordal graphs. For more detailed information on clique trees, see [2].

## 3 An algorithm for chordal graphs

In this section, we show that we can find a minimum height elimination tree for a chordal graph $G$ by using a clique tree of $G$. Since minimum height elimination tree orderings are not always minimum fill orderings, we will not be producing perfect elimination orderings in general. Thus we are interested in finding minimum height elimination trees without restricting the amount of fill-in. The recursive algorithm presented in this section will not be of polynomial-time complexity, but will serve to illustrate how clique trees can be used for this purpose. Our approach is as follows: Let $G$ be a chordal graph. Solve the following two problems.

1. Find a clique tree $\mathcal{T}$ of $G$.

2. Using $\mathcal{T}$, find a minimum height elimination tree for $G$.

A linear-time algorithm for solving Problem 1 can be found in [2]. In this section, we show how Problem 2 can be solved and give an algorithm for this. From Theorem 2.1, we know that there always exists a nested dissection ordering with minimal separators, that results in a minimum height elimination tree for $G$. As we have seen in the previous section, the edges in a clique tree $\mathcal{T}$ of $G$ are the minimal $v$-$w$ separators in $G$. Since every minimal separator in $G$ is also a minimal $v$-$w$ separator for some $v$ and $w$ in $G$, $\mathcal{T}$ provides all the necessary information about every possible minimal separator in $G$. The idea behind our algorithm is to exploit this property of $\mathcal{T}$. A minimum height elimination tree for $G$ can then be found by trying out every minimal separator $S$ as the top separator, and finding minimum height elimination trees for the components in $G - S$ recursively.

Before we give the algorithm, we show in the next lemma that a clique tree $\mathcal{T}$ of $G$ provides information not only about the minimal separators in $G$, but also about the minimal separators in every induced subgraph of $G$.

LEMMA 3.1. *Let $G$ be a chordal graph and $\mathcal{T}$ a clique tree of $G$. Let $H$ be any induced subgraph of $G$. Any minimal $v$-$w$ separator $S_H$ in $H$ is a subset of at least one edge in $\mathcal{T}$.*

PROOF. Since induced subgraphs of chordal graphs are also chordal, $H$ is a chordal graph. Let $S_H$ be a minimal $v$-$w$ separator in $H$. By Lemma 2.2, $S_H$ is a clique. Since $S_H$ is a minimal $v$-$w$ separator, by Lemma 2.8 there exist at least two maximal cliques $C_1$ and $C_2$ in $H$ such that $S_H = C_1 \cap C_2$. Now, $C_1 \subseteq K_1$ and $C_2 \subseteq K_2$ for two distinct maximal cliques $K_1$ and $K_2$ in $G$. To see this, remember that $H$ is induced and $C_1$ and $C_2$ cannot be subsets of one single maximal clique in $G$, since there are vertices in $C_1$ and $C_2$ that are not neighbors. Clearly, $K_1$ and $K_2$ are vertices in the clique tree $\mathcal{T}$ and $S_H \subseteq K_1 \cap K_2$. Thus, by Corollary 2.7, $S_H$ is a subset of each edge on the path between $K_1$ and $K_2$ in $\mathcal{T}$. $\square$

As we can see by Lemma 3.1, the clique tree provides all the information that is needed by each step of a nested dissection algorithm. Given a chordal graph $G$ and a clique tree $\mathcal{T}$ of $G$, the minimal separators in each subgraph that we must consider, are edges in $\mathcal{T}$.

Before we can continue, we also need to show the relationship between subtrees of $\mathcal{T}$ and induced subgraphs of $G$. Let $S$ be a minimal $v$-$w$ separator in $G$ and let $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ be a clique tree of $G$. We define $\mathcal{T} - S$ to be the forest obtained by removing the vertices in $S$ from every maximal clique and every edge in $\mathcal{T}$. The edges and maximal cliques that become empty are removed from $\mathcal{T}$. Since there is at least one edge $(C_i, C_j) = S$ in $\mathcal{T}$, $\mathcal{T} - S$ is disconnected. Let $\mathcal{T}' = (\mathcal{V}', \mathcal{E}')$ be a component of $\mathcal{T} - S$ with $\mathcal{V}' = \{C'_{i_1}, ..., C'_{i_k}\}$, and $\mathcal{E}' = \{E'_{i_1}, ..., E'_{i_{k-1}}\}$. Then $\mathcal{T}'$ corresponds to a connected induced subtree $\mathcal{T}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ of $\mathcal{T}$ with $\mathcal{V}_1 = \{C_{i_1}, ...C_{i_k}\}$ and $\mathcal{E}_1 = \{E_{i_1}, ..., E_{i_{k-1}}\}$, where $C'_{i_j} \subseteq C_{i_j}$ for $1 \leq j \leq k$, and $E'_{i_j} \subseteq E_{i_j}$ for $1 \leq j < k$.

Let $G' = (V', E')$ be a component of $G - S$ such that $V' = \{v \mid v \in C'_{i_j}, 1 \leq j \leq k\}$. Such a component must exist by Theorem 2.4. Notice that $\mathcal{T}'$ is not necessarily a proper clique tree for $G'$, since there may be cliques in $\mathcal{T}$ that no longer are maximal after the removal of the vertices in $S$. But as we have seen in Lemma 3.1, minimal separators in $G'$ are subsets of minimal separators in $G$. They will therefore be edges of $\mathcal{T}'$, which thus includes all the information we need about the minimal separators of $G'$. By Theorem 2.6, a clique that is not maximal in $\mathcal{T}'$ must be a subset of one of its neighbors. A proper clique tree for $G'$ can be found by merging each clique in $\mathcal{T}'$ that is not maximal with its neighbor that is a superset.

The next theorem shows that we can find minimum height elimination trees for chordal graphs, and an algorithm follows from the proof of the theorem.

THEOREM 3.2. *We can find a minimum height elimination tree for a chordal graph $G$, given a clique tree $\mathcal{T}$ of $G$.*

PROOF. The proof is by induction on the number of maximal cliques. Our induction hypothesis is the following.

*Induction hypothesis:* We can find a minimum height elimination tree for $G$ if $\mathcal{T}$ contains $\leq m$ maximal cliques.

The base case is when $m = 1$. Then the clique tree contains one maximal clique $C$, which may be empty. Thus we cannot have branching, and the minimum elimination tree height for $G$ is 0 if $C$ is empty, and $|C| - 1$ otherwise.

Assume that the induction hypothesis is true for $m = k$, where $k \geq 1$, and we will now show that it is true for $m = k+1$. Let $\mathcal{T}$ be a clique tree of a chordal graph having $k+1$ maximal cliques for a chordal graph $G$. Let $S$ be an edge in $\mathcal{T}$ that is a minimal separator in $G$. By the induction hypothesis, for each component $\mathcal{T}'$ in $\mathcal{T} - S$, we know how to find a minimum height elimination tree for the subgraph of $G$ corresponding to $\mathcal{T}'$. We find an elimination tree $T$ for $G$ in the following way: $S$ is the top separator in $T$, and for each $\mathcal{T}'$, $S$ has a minimum height elimination tree of $\mathcal{T}'$ as a child. In order to find the height of $T$, we add $|S|$ to the height of the highest child of $S$. However, $T$ is not necessarily a minimum height elimination tree for $G$. In order to find a minimum height elimination tree, we repeat the process described above for each edge $S$ in $\mathcal{T}$ that is a minimal separator in $G$. $\square$

An algorithm can now be designed directly from this proof. For simplicity, we give an algorithm that computes only the minimum elimination tree height, but the actual elimination tree can also be computed in the same way. Algorithm *MinTree*, which is shown in Figure 3.1, returns the minimum elimination tree height for a chordal graph $G$, given a clique tree $\mathcal{T}$ of $G$.

In the proof of Theorem 3.2, we considered only the minimal separators of $G$. Since the edges in $\mathcal{T}$ represent all the minimal $v$-$w$ separators in $G$, and every minimal separator is a minimal $v$-$w$ separator for some $v$ and $w$ in $G$, there is more information in $\mathcal{T}$ than we need. As long as every minimal separator is considered, we see from Theorem 2.1 that a minimum height elimination tree is found. So the correct result is found by Algorithm *MinTree* although we also consider the edges in $\mathcal{T}$ that are not minimal separators. In fact, the minimum height elimination tree found by the algorithm may contain separators that are not minimal.

In Algorithm *MinTree*, the proper clique tree for each component $\mathcal{T}'$ is not computed. By the same argument as above, this is not necessary since each $\mathcal{T}'$ includes all the information we need and more.

The time complexity of Algorithm *MinTree* is exponential in the number of maximal cliques in $G$. Thus, this algorithm is not practical for large graphs. However, *MinTree* is useful for theoretical reasons. It is simple and easy to understand and illustrates the idea of exploiting the properties of clique trees to find minimum elimination tree

```
Algorithm MinTree (T: Clique tree): integer;

begin
    if T is empty then
        MinTree = 0
    else
        if T has only one maximal clique C then
            MinTree = |C| - 1
        else
            min = ∞;
            for each edge S in T do
                max = 0;
                for each component T' in T - S do
                    height = MinTree(T');
                    if max < height then
                        max = height;
                end-for;
                if min > |S| + max then
                    min = |S| + max;
            end-for;
            MinTree = min;
        end-if;
end;
```

Figure 3.1: Algorithm *MinTree*.

height for chordal graphs. The technique used to develop this algorithm will help us to design polynomial-time algorithms for subclasses of chordal graphs. Such an algorithm for a subclass of chordal graphs, which we call $\Omega$, is presented in the next section. Our main goal is to design an efficient algorithm for interval graphs. The results in the next section will build a bridge from general chordal graphs to interval graphs, and illustrate how dynamic programming can be used to produce polynomial-time algorithms.

## 4 The class $\Omega$

Our aim now is to find a class of graphs for which an algorithm similar to $MinTree$ finds the minimum elimination tree height in polynomial time. Since $MinTree$ is designed for chordal graphs, it is natural to consider a subclass of chordal graphs. We now give a formal definition of a subclass of chordal graphs, which we call $\Omega$. We will see from the results in this section that $\Omega$ fits our purposes.

DEFINITION 4.1. *A graph $G$ belongs to $\Omega$ if $G$ is chordal and has at most two maximal cliques that contain simplicial vertices.*

As we can see from this definition, a graph $G \in \Omega$ is either a complete graph, or a chordal graph with exactly two maximal cliques that contain simplicial vertices. We consider the graphs in $\Omega$, and we show that we can find minimum height elimination trees for these graphs in polynomial time by using clique trees. Our approach is the same as in the previous section. As we have seen, by trying out every ordering of the separators in a graph $G \in \Omega$, we can find a minimum height elimination tree for $G$. However, by exploiting some properties of the graphs in $\Omega$ and their clique trees, we can restrict the amount of work, so that we can find a minimum height elimination tree in polynomial time for the graphs in $\Omega$.

We now present some results about the graphs in $\Omega$ and their clique trees, which we will use to develop an algorithm similar to $MinTree$.

LEMMA 4.1. *A graph $G \in \Omega$ if and only if every perfect elimination ordering on $G$ results in an elimination tree that has either one or two leaves.*

PROOF. *(if)* If $G$ has more than two maximal cliques with simplicial vertices, then there are at least three non-adjacent simplicial vertices in $G$. Thus these non-adjacent simplicial vertices can be eliminated simultaneously without introducing any fill-in. This however, results in an elimination tree that has at least three leaves.

*(only if)* If a perfect elimination ordering on $G$ results in an elimination tree with more than two leaves, then there must exist at least three vertices in $G$ that can be eliminated simultaneously. This is possible only when $G$ has at least three non-adjacent simplicial vertices. But then $G$ must have at least three maximal cliques with simplicial vertices. □

We now define a special type of graph that will be used throughout the rest of this paper.

DEFINITION 4.2. *A trail is a tree $T = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$, and $E = \{(v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n)\}$.*

We continue by showing in Theorem 4.3 that a graph $G \in \Omega$ has a unique clique tree that is a trail. The following lemma will help us to prove Theorem 4.3.

LEMMA 4.2. *Let $G$ be a chordal graph, and let $\mathcal{T}$ be a clique tree of $G$. Then every leaf clique in $\mathcal{T}$ contains at least one vertex $v$, where $v$ is a simplicial vertex in $G$.*

PROOF. Let $C$ be a leaf clique in $\mathcal{T}$, and let $C'$ be the only neighbor of $C$ in $\mathcal{T}$. Thus the edge $(C, C')$ is the only edge incident to $C$ in $\mathcal{T}$. Let $\mathcal{G}$ be the clique graph of $G$. For every neighbor $C_i$ of $C$ in $\mathcal{G}$, we have from Corollary 2.7 that $(C_i, C) \subseteq (C, C')$. Thus, the vertices in $C - (C, C')$ do not belong to any other maximal cliques than $C$, and are simplicial vertices in $G$. Since $C \neq C'$, there must be at least one vertex in $C - (C, C')$. □

THEOREM 4.3. *Let $G$ be a graph in $\Omega$. Then $G$ has a unique clique tree $\mathcal{T}$ that is a trail.*

PROOF. Let $\mathcal{T}$ be a clique tree of $G$. By the definition of $\Omega$ and Lemma 4.2, $\mathcal{T}$ has at most two leaf cliques. If $G$ has only one maximal clique, then this is the only clique in $\mathcal{T}$, and $\mathcal{T}$ is therefore unique. If $G$ has two or more maximal cliques, then any clique tree of $G$ must have at least two leaf cliques. Since $G \in \Omega$, it follows that $\mathcal{T}$ has exactly two leaf cliques, and the leaf cliques are the two maximal cliques in $G$ with simplicial vertices. Therefore $\mathcal{T}$ must be a trail. We now show by induction that $\mathcal{T}$ must be unique.

*Induction hypothesis:* For every graph $G \in \Omega$ with $\leq m$ maximal cliques, there exists a unique clique tree $\mathcal{T}$.

The base case where $m = 3$ is the first interesting case since all trees with two vertices are isomorphic. Let $G$ have 3 maximal cliques $C_1, C_2$ and $C_3$, where $C_1$ and $C_3$ are the ones with simplicial cliques. Then in a clique tree $\mathcal{T}$, $C_1$ and $C_3$ must be leaf cliques and $C_2$ must have edges to $C_1$ and $C_3$. This is the only possible clique tree of $G$ and thus $\mathcal{T}$ is unique.

Let the induction hypothesis be true for $m = k$, where $k \geq 3$, and we show that it is true for $m = k + 1$. Let $G$ be a graph in $\Omega$ with $k + 1$ maximal cliques, and let $C_1$ and $C_{k+1}$ be the two maximal cliques in $G$ with simplicial vertices. As we have mentioned before, in every clique tree of $G$, $C_1$ and $C_{k+1}$ must be leaf cliques. Let $\mathcal{T}$ be a clique tree of $G$; we show that $\mathcal{T}$ is unique. Let $C_1, C_2$ and $C_3$ be maximal cliques in $\mathcal{T}$ as shown in Figure 4.1 a). Let $G'$ be an induced subgraph of $G$ that we get by taking away the simplicial vertices in $C_1$, and let $\mathcal{T}'$ be the remaining clique tree when the same vertices are removed from $\mathcal{T}$. Then $\mathcal{T}'$ is a clique tree of $G'$. By Corollary 2.7, in $\mathcal{T}$, $(C_1, C_2)$ must contain every vertex that $C_1$ has in common with other maximal cliques. Thus when the simplicial vertices in $C_1$ are removed, the remaining vertices of $C_1$ make a subset of $C_2$. Therefore, the remaining clique tree $\mathcal{T}'$ has $k$ maximal cliques as shown in Figure 4.1 b). We know that if $G' \in \Omega$, then by the induction hypothesis, $\mathcal{T}'$ is unique. Therefore, we now show that $G' \in \Omega$. In order to show this, we need to verify the following two statements:

1. $C_2$ has simplicial vertices in $G'$.

2. No other maximal clique $C_i$, where $2 < i < k + 1$ has simplicial vertices in $G'$.

In order to show Statement 1, we note that in $\mathcal{T}$, $(C_1, C_2)$ must contain at least one vertex $v$ that is contained in no other maximal clique than $C_1$ and $C_2$. Otherwise, if every vertex in $(C_1, C_2)$ were contained in some other maximal clique, then by Corollary 2.7 $(C_2, C_3)$ would contain $(C_1, C_2)$. But since $C_2$ has no simplicial vertices in $\mathcal{T}$, this would mean that $C_2 \subseteq C_3$ and lead to a contradiction. Thus $v$ becomes a simplicial vertex in $C_2$ in $G'$. We now show Statement 2. In order for a maximal clique $C_i, 3 \leq i \leq k$, to have simplicial vertices in $\mathcal{T}'$, $C_i$ must have vertices in common with $C_1$ in $\mathcal{T}$. Otherwise, the removal of the simplicial vertices in $C_1$ would not affect $C_i$. But since $(C_1, C_2)$ contains $C_1 \cap C_i$, every vertex in $C_1 \cap C_i$ is also contained in $C_2$ and cannot be a simplicial vertex. Thus, $\mathcal{T}'$ is unique, and $\mathcal{T}$ must also be unique. ◻

Graphs that do not belong to $\Omega$ may also have clique trees that are trails. One of the several clique trees for a chordal graph $G$ may be a trail although $G$ does not belong to $\Omega$. Also, a graph $G \notin \Omega$ may have different clique trees that are all trails.

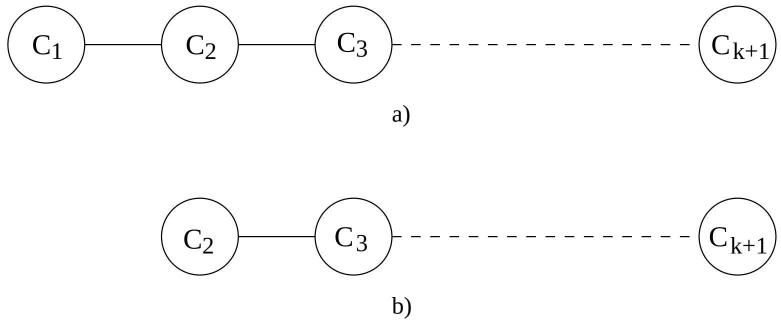


Figure 4.1: $\mathcal{T}$ and $\mathcal{T}'$ in the proof of Theorem 4.1.

Let $G$ be a graph in $\Omega$. Theorem 4.3 states that $G$ has a unique clique tree $\mathcal{T}$ that is a trail. Any perfect elimination ordering on $G$ must start by eliminating simplicial vertices. Since the leaf cliques in $\mathcal{T}$ are the only maximal cliques in $G$ with simplicial vertices, a perfect elimination ordering can first eliminate the simplicial vertices in the leaf cliques. Let one of the leaf cliques in $\mathcal{T}$ be numbered 1. A perfect elimination ordering $\alpha$ on $G$ can be found in the following way: Eliminate first the simplicial vertices in $C_1$. As we have seen in the proof of Theorem 4.3, when the simplicial vertices in $C_1$ are eliminated, the only neighbor of $C_1$ becomes a leaf clique with simplicial vertices. We call this neighbor $C_2$, and continue by eliminating the simplicial vertices in $C_2$. Whenever the simplicial vertices in a leaf clique $C_i$ are eliminated, the only neighbor of $C_i$ becomes a leaf clique with simplicial vertices in the remaining clique tree. We can then number this neighbor as $C_{i+1}$, and continue by eliminating the simplicial vertices in $C_{i+1}$. Let us start with $C_1$ and repeat this process until the remaining clique tree is empty. This elimination ordering $\alpha$ results in an elimination tree with one leaf, and implies a natural linear ordering of the maximal cliques in $G$.

In the rest of this section, we assume that the maximal cliques of $G$ are numbered from 1 to $m$ as described above. This argument can also be applied to both ends of the clique tree, thus any perfect elimination ordering must proceed from the ends of the clique tree to the interior. By Lemma 2.3, any maximal clique on the path from $C_1$ to $C_m$ can be eliminated last in a perfect elimination ordering.

This discussion about perfect elimination orderings is included here only in order to define a natural linear ordering of the cliques in $\mathcal{T}$. As mentioned before, the orderings that minimize the elimination tree height, are generally not perfect elimination orderings. Thus the algorithms we present in this paper do not generally produce perfect elimination orderings. We now continue with some more results that will help us to show how $\mathcal{T}$ can be used to find a minimum height elimination tree for $G$. The number of maximal cliques in $G$ will always be denoted by $m$, unless specified otherwise.

LEMMA 4.4. *Let $G$ be a graph in $\Omega$. Then every minimal $v$-$w$ separator in $G$ divides $G$ into exactly two components.*

PROOF. The proof is by contradiction. Assume that there exists a minimal $v$-$w$

separator $S$ in $G$ that divides $G$ into at least three components. By Lemma 2.2, $S$ is a clique, and by Lemma 2.3, there exists a perfect elimination ordering $\alpha$ that eliminates $S$ last. Then in each of the three components, there must be simplicial vertices such that each component can be eliminated without introducing fill-in. This implies the existence of at least three maximal cliques in $G$ with simplicial vertices, and thus contradicts the fact that $G \in \Omega$. □

LEMMA 4.5. *Let $G$ be a graph in $\Omega$ and let $\mathcal{T}$ be the clique tree of $G$. Then there do not exist two distinct edges $S$ and $S'$ in $\mathcal{T}$, such that $S' \subseteq S$.*

PROOF. Suppose there exist two distinct edges $S$ and $S'$ for which $S' \subseteq S$. Let $S = (C_1, C_2)$, and $S' = (C_1', C_2')$. Without loss of generality assume the cliques occur in the following order:

$$..., C_1, C_2, ..., C_1', C_2', ....$$

Possibly, $C_2 = C_1'$. Consider the tree obtained by removing the edge $(C_1', C_2')$, and adding the edge $(C_2', C_1)$. Since $S' \subseteq S$, the new tree has weight no smaller than that of the original; moreover, the weight of the new tree cannot be larger, else the original tree would not be a clique tree. It follows that the new tree is a clique tree of $G$. The clique $C_1'$ is a leaf in the new tree and hence by Lemma 4.2, $C_1'$ has simplicial vertices. With at least three cliques with simplicial vertices, $G$ cannot belong to $\Omega$. □

The following corollary follows directly from Lemma 4.5.

COROLLARY 4.6. *Let $G$ be a graph in $\Omega$, and let $\mathcal{T}$ be the clique tree of $G$. Then every separator represented by the edges in $\mathcal{T}$, is a minimal separator in $G$.*

To be able to use nested dissection with minimal separators to find a minimum height elimination tree, we must find the minimal separators in $G$. We know from Lemma 2.8 that a clique tree for any chordal graph contains information about every minimal $v$-$w$ separator in the graph. We have mentioned before that every minimal separator is also a minimal $v$-$w$ separator for some $v$ and $w$ in $G$. Thus, clique trees in general, give us more separators than we want, since we are only interested in the minimal separators. For the graphs in $\Omega$ however, we see by Corollary 4.6 that the separators provided by the clique trees, are exactly the minimal separators that we need.

DEFINITION 4.3. *Let $G$ be a graph in $\Omega$, and $\mathcal{T}$ the clique tree of $G$. For all $i, j$, where $1 \leq i \leq j \leq m$, we define $H_{ij}$ to be the subgraph of $G$ induced by the set of vertices internal to maximal cliques $C_i, C_{i+1}, ..., C_j$. In other words, $H_{ij}$ is the graph induced by the vertices in the following set: $\{v \mid v \in C_k$ for some $k$ such that $1 \leq i \leq k \leq j$, and $v \notin C_k$ for $k < i$ or $k > j\}$.*

LEMMA 4.7. *Let $G$ be a graph in $\Omega$, and $\mathcal{T}$ the clique tree of $G$. For all $i, j$, where $1 \leq i \leq j \leq m$, $H_{ij} \in \Omega$.*

PROOF. Let $1 < i \leq j < m$ (the cases where either $i = 1$ or $j = m$ will follow readily from the proof). We let $S_k$ denote the separator $(C_k, C_{k+1})$, for $1 \leq k < m$. By Lemma 4.4, removing $S_{i-1}$ from $G$ divides the remaining graph into two connected components. These components are $H_{1,i-1}$ and $H_{im}$. We will first prove by contradiction that $H_{im}$ belongs to $\Omega$. Assume that $H_{im} \notin \Omega$. Then $H_{im}$ must have at least three maximal cliques with simplicial vertices. Let $i'$ be the least integer greater than $i$ such that $C_{i'} - S_{i-1}$ is a maximal clique in $H_{im}$. Let $C_k' = C_k - S_{i-1}$, $k = i', \ldots, m$. By Lemma

4.2, the leaf cliques $C'_{i'}$ and $C'_m$ of $H_{im}$ contain simplicial vertices. Additionally, by our assumption, there must exist a maximal clique $C'_l$, where $i' < l < m$, such that $C'_l$ has a simplicial vertex $x$ in $H_{im}$. A simplicial vertex can only occur in one maximal clique, so the adjacency set of $x$ in $H_{im}$ is the clique $C'_l - \{x\}$. Now, either $x$ is simplicial (belonging only to $C_l$) also in $G$, or $S_{i-1}$ contains some neighbors of $x$, whose removal makes $x$ a simplicial vertex. In the latter case, $x$ must belong to every $C_k$, for $i \leq k \leq l$. This contradicts the fact that $x \notin C'_{i'}$. Therefore, $x$ must be simplicial also in $G$, and since $G \in \Omega$, we have arrived at a contradiction. Thus $H_{im}$ belongs to $\Omega$. (It can be shown similarly that $H_{1,i-1}$ also belongs to $\Omega$.) To show that $H_{ij} \in \Omega$, assume that $i' \leq j$. (If $j < i'$, then $H_{ij}$ is empty and thus belongs to $\Omega$.) We remove $S'_j = S_j - S_{i-1}$ from $H_{im}$, dividing the remaining graph into the two connected components $H_{ij}$ and $H_{j+1,m}$. By the same argument as above, $H_{ij}$ (and $H_{j+1,m}$) belong to $\Omega$. □

By Lemma 4.7, the results that we have shown on $G$, where $G$ is a graph in $\Omega$, are also true for $H_{ij}$. This makes it easier to design a recursive algorithm, since we know that the subgraphs that we work on in each step have the same properties as the original graph $G$. We have also mentioned in the previous section, that every minimal separator that needs to be considered in each step is represented in the original clique tree $\mathcal{T}$ of $G$. With this background we are now ready to design an algorithm for finding minimum elimination tree height for graphs in $\Omega$.

As we will see in the proof of the following theorem, an elimination tree of minimum height for a graph $G \in \Omega$ can be found without having to try every permutation of the vertices in $G$. An algorithm for finding a minimum height elimination tree follows directly from this proof.

THEOREM 4.8. *Let $G$ be a graph in $\Omega$, and let $\mathcal{T}$ be the clique tree of $G$. We can find a minimum height elimination tree for $G$ by using only the information provided by $\mathcal{T}$.*

PROOF. The proof is by induction, and our induction hypothesis is as follows.

*Induction hypothesis:* We know how to find an elimination tree of minimum height for $H_{ij}$, where $(j - i) \leq k$.

We show this by induction on $k$. The base case is when $k = 0$. This means that $H_{ij}$ is either empty or a complete graph ($H_{11}$ and $H_{mm}$ are complete graphs, whereas $H_{ii}$ for $1 < i < m$ is empty). If $H_{ij}$ is empty then the elimination tree height is 0. Otherwise, the elimination tree height is one less than the number of the vertices in $H_{ij}$, and the elimination tree consists of the vertices in $H_{ij}$, and has only one leaf.

We assume now that the induction hypothesis is true, and show how this enables us to find a minimum height elimination tree for $H_{ij}$, where $(j - i) = k + 1$. Let $\mathcal{T}'$ be the subtree of $\mathcal{T}$ that corresponds to $H_{ij}$. We choose an edge in $\mathcal{T}'$ to be the top separator in the elimination tree. Let $S = (C_p, C_q)$ be this edge, where $C_p$ and $C_q$ are two consecutive maximal cliques in $\mathcal{T}'$. By Lemma 4.4, the removal of $S$ divides $H_{ij}$ into two components $H_{ip}$ and $H_{qj}$. By the induction assumption, since $(p - i) \leq k$ and $(j - q) \leq k$, we can find minimum height elimination trees for $H_{ip}$ and $H_{qj}$. Let $T_{min}(H_{ij})$ denote a minimum height elimination tree for $H_{ij}$. Further, let $T_S(H_{ij})$ be the elimination tree of $H_{ij}$ with $S$ as the top separator, and $T_{min}(H_{ip})$ and $T_{min}(H_{qj})$

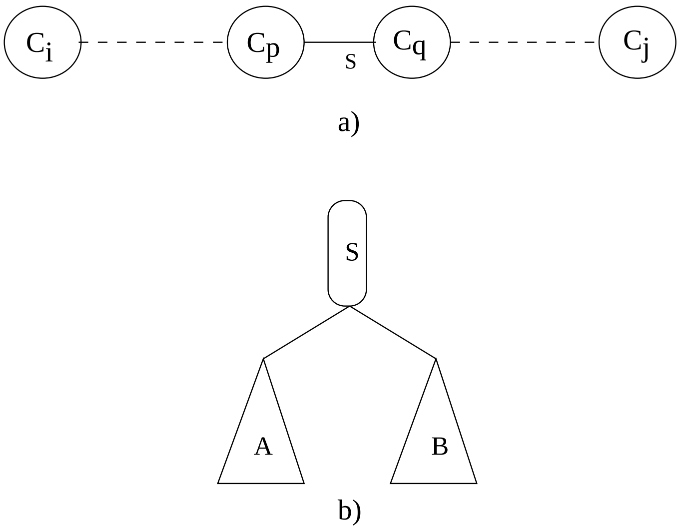


Figure 4.2: a) $\mathcal{T}'$. b) $T_S(H_{ij})$, where $A = T_{min}(H_{ip})$, and $B = T_{min}(H_{qj})$.

as the two subtrees that hang from $S$. This is illustrated in Figure 4.2. Then

$$Height(T_S(H_{ij})) = |S| + \max(Height(T_{min}(H_{ip})), Height(T_{min}(H_{qj}))).$$

In order to find a minimum height elimination tree for $H_{ij}$, we find $T_S(H_{ij})$ for every edge $S$ in $H_{ij}$, as described above. Then we choose the elimination tree that has the minimum height. This way, every minimal separator in $H_{ij}$ is tried as the top separator, and the one that gives the minimum height, is chosen. Hence a minimum height elimination tree is found for $H_{ij}$, since there always exists a nested dissection ordering with minimal separators that results in a minimum height elimination tree. □

We now present Algorithm *Height* that computes the minimum elimination tree height for a graph in $\Omega$ as described in the proof of Theorem 4.8. For simplicity, we compute only the height of the elimination tree and not the tree itself. Let $C_1, ..., C_m$ be the maximal cliques in $\mathcal{T}$, and let $S_1, ..., S_{m-1}$ be the edges in $\mathcal{T}$, where $S_i = (C_i, C_{i+1})$. The algorithm returns the height of a minimum height elimination tree for $H_{fromc,toc}$, where $fromc \leq toc \leq m$. Algorithm *Height* is shown in Figure 4.3.

THEOREM 4.9. *Let $G$ be a graph in $\Omega$, and let $\mathcal{T}$ be the clique tree of $G$. Then $Height(\mathcal{T}, 1, m)$ is the minimum elimination tree height for $G$.*

PROOF. Follows directly from the proof of Theorem 4.8 and Algorithm *Height*.

Although Algorithm *Height* produces the desired result, it is not efficient. There are parts that are redundant and many subresults are computed several times. In order to achieve polynomial time complexity, we now present another algorithm, which is called *DynHeight*, that solves the same problem by using dynamic programming. We use a two dimensional array *Opt* $[1..m, 1..m]$ to store the optimal elimination tree heights of the subgraphs of $G$. The number *Opt* $[i, j]$ contains the height of a lowest elimination tree for $H_{ij}$. Algorithm *DynHeight* is shown in Figure 4.4.

**Algorithm** *Height* ($\mathcal{T}$: Clique tree; *fromc, toc:* integer): integer;

**begin**
  $A = \{v \in C_j \mid j < fromc \text{ or } j > toc\}$;
  **for** i = *fromc* **to** *toc* − 1 **do**
    $C'_i = C_i - A$;
    $S'_i = S_i - A$;
  **end-for;**
  $C'_{toc} = C_{toc} - A$;
  **if** $fromc = toc$ **then**
    **if** $C'_{fromc}$ is empty **then**
      $Height = 0$
    **else**
      $Height = |C'_{fromc}| - 1$;
  **else**
    opt = $\infty$;
    **for** i = *fromc* **to** *toc* − 1 **do**
      min = $|S'_i| + max\ (Height(\mathcal{T}, fromc, i), Height(\mathcal{T}, i+1, toc))$;
      **if** min < opt **then**
        opt = min;
    **end-for;**
  **end-if;**
  *Height* = opt;
**end;**

Figure 4.3: Algorithm *Height*.

As we can see in Algorithm *DynHeight*, when we compute *Opt* $[i, j]$ we need to know *Opt* $[i, k]$ and *Opt* $[k+1, j]$ for $i \le k < j$. We compute and use only the upper triangular part of the matrix *Opt*, since we only consider $H_{ij}$ where $i \le j$. Thus before computing *Opt* $[i, j]$, we must have computed rows $i+1, i+2, ..., m$, and elements $i, i+1, ..., j-1$ in row $i$. This explains the structure of the loops in the algorithm. It is then easy to see that the entry in *Opt* $[1, m]$ computed by $DynHeight(\mathcal{T}, Opt)$ is the same as the result returned by $Height(\mathcal{T}, 1, m)$.

In order to find the time complexity of *DynHeight*, we note the following: computing $A$ and updating the current maximal cliques can be done in $O(n)$ steps although we have presented a less efficient way of doing this in our algorithm in order to make it easier to understand. (The set $A$ can also be expressed in terms of items within the clique tree structure: $A = S_{i-1} \cup S_j$, where $2 \le i \le j \le m$.) Thus the total amount of

```
Algorithm DynHeight (T: Clique tree; Opt: Array of integers);

begin
    for i = 1 to m do
        for j = 1 to m do
            Opt [i,j] = ∞;
    for i = m downto 1 do
        for j = i to m do
            A = {v ∈ C_k | k < i or k > j};
            for k = i to j − 1 do
                C'_k = C_k − A;
                S'_k = S_k − A;
            end-for;
            C'_j = C_j − A;
            if i = j then
                if C'_i is empty then
                    Opt [i,j] = 0
                else
                    Opt [i,j] = |C'_i| − 1;
            else
                for k = i to j −1 do
                    min = |S'_k| + max (Opt [i,k], Opt [k+1,j]);
                    if min < Opt [i,j] then
                        Opt [i,j] = min;
                end-for;
        end-for;
end;
```

Figure 4.4: Algorithm *DynHeight*.

work needed to be done is:

$$O(m^2) + \sum_{i=1}^{m}\sum_{j=i}^{m}(O(n) + O(j-i)) = O(m^2) + O(nm^2) + O(m^3) = O(nm^2).$$

This gives us a time complexity of $O(nm^2)$. Since $n \geq m$ in a chordal graph, this time complexity is no worse than $O(n^3)$.

It should be pointed out that when designing *DynHeight* we have concentrated on simplicity and tried to make it easy to understand. Our main goal was merely to achieve polynomial time complexity; it was not to design an algorithm as efficient as possible. Therefore, it might be possible to reduce the time bound on *DynHeight* by implementing it in a different way than we have done here. The algorithm may also be implemented in parallel to get a better time complexity.

We can find the actual elimination tree by slightly modifying this algorithm. This can be done by using another two dimensional array *Root* $[1..m,1..m]$ to store the s-nodes in a separator tree for $G$ corresponding to a minimum height elimination tree. The root separator in the separator tree for $H_{ij}$ is stored in *Root* $[i,j]$. This way, the separator tree corresponding to a minimum height elimination tree can be found. We have to initialize the array *Root* to be empty in the beginning of the algorithm. The updating of the array *Opt*, which is

```
if min < Opt [i,j] then
    Opt [i,j] = min;
```

must be changed to

```
if min < Opt [i,j] then
    Opt [i,j] = min;
    Root [i,j] = S'_k;
end-if;
```

With this modification, the algorithm is able to provide the information needed to construct a minimum height elimination tree for $G$. The idea of dynamic programming will be used in the next section to present a polynomial-time algorithm for an important superset of $\Omega$.

## 5 Interval graphs

In Section 3, we showed that *any* clique tree of a chordal graph $G$ could be used to find a minimum height elimination tree for $G$. In Section 4, we showed that if the clique tree is *unique* and a trail, where *no* interior clique has simplicial vertices, then dynamic programming can be used to develop a polynomial-time algorithm. In this section, we combine these two results and show how to find minimum height elimination trees in polynomial time for graphs that have *a* clique tree that is a trail *without any restrictions* on simplicial vertices. Such a class will obviously be a superclass of $\Omega$ and a subclass of chordal graphs.

By Theorem 3.2 any clique tree of $G$ can be used as long as we try every possible ordering of the minimal separators in $G$. If there exists a clique tree $\mathcal{T}$ for $G$ that is a trail, then the maximal cliques in $G$ can be numbered from left to right as they appear in $\mathcal{T}$. This gives a linear ordering of the maximal cliques in $G$, and thus dynamic programming can easily be used. It turns out that the class of graphs that have this property is indeed the well-known class called interval graphs. We first give a definition of this class.

DEFINITION 5.1. *A graph $G = (V,E)$ is an interval graph if there is a mapping $I$ of the vertices in $G$ into sets of consecutive integers such that for each pair of vertices $v,w \in V$ the following is true: $(v,w) \in E \Leftrightarrow I(v) \cap I(w) \neq \emptyset$.*

The following theorem, here stated in terms of clique trees, was first proved by Gilmore and Hoffman [12] (see also [13], page 172).

THEOREM 5.1. *A graph $G = (V, E)$ is an interval graph if and only if there exists a clique tree $\mathcal{T}$ for $G$, where $\mathcal{T}$ is a trail.*

Booth and Lueker give a linear-time algorithm for testing whether a graph is an interval graph in [4]. Their algorithm is constructive so that the set $I(v)$ for each vertex $v$ in $G$ is found in linear time and thus is of linear cardinality. Therefore the construction of a clique tree that is a trail can be done efficiently.

Let $G$ be an interval graph, $S$ a minimal separator in $G$, and $\mathcal{T}$ a clique tree of $G$ that is a trail. We now present Algorithm *Interval* that finds minimum elimination tree height for $G$. We will use the same method as in Algorithm *DynHeight*, but this time a few more things have to be checked. Let $I_{ij}$ be defined for interval graphs in the same way as $H_{ij}$ was defined for the graphs in $\Omega$. As we have done in Algorithm *DynHeight*, we organize the computation such that when we are working with $I_{ij}$, the minimum elimination tree height for each $I_{kl}$, $i \leq k \leq l \leq j$, is already computed.

Golumbic shows in [13] that the interval graph property is hereditary. Thus each $I_{ij}$ is also an interval graph. The difference between subgraphs $I_{ij}$ and $H_{ij}$ is that each $I_{ij}$ is not necessarily connected, but the connected components of $I_{ij}$ are interval graphs. Since by Theorem 2.4 the connected induced subgraphs of $G$ correspond to connected induced subtrees of $\mathcal{T}$, we need first to find out if the subtree $\mathcal{T}'$ of $\mathcal{T}$ that corresponds to $I_{ij}$ is connected. If not, $I_{ij}$ cannot have an elimination tree. In this case, by the way our algorithm is organized, the minimum height elimination trees for the connected components of $I_{ij}$ are already computed. If $I_{ij}$ is connected, then we can proceed in the same way as in *DynHeight* by trying out each edge on the path from $C'_i$ to $C'_j$ in $\mathcal{T}'$ as the top separator for $I_{ij}$. But each such edge $S'_k$ can divide $I_{ij}$ into more than two components. Thus we have to find the components in $\mathcal{T}' - S'_k$ and check their minimum elimination tree height, which already are computed and can be fetched from an array $Opt[i, j]$. Algorithm *Interval* is shown in Figure 5.1. The maximal cliques and the edges in $\mathcal{T}$ are assumed to be numbered as for Algorithm *DynHeight*.

Like Algorithm *DynHeight*, this algorithm finds the minimum elimination tree height, and not the actual elimination tree. But the elimination tree can be found by modifying the algorithm slightly, as explained in connection with Algorithm *DynHeight* in Section 4.

In Algorithm *Interval* we have introduced a new loop in order to find the components of $I_{ij} - S'_k$. This increases the time complexity of the algorithm by a factor of $n$, giving $O(n^4)$ for *Interval*. Again we would like to stress that the algorithms are given to illustrate the results presented in this paper. Thus comprehensibility is given the highest priority, and the algorithms most probably can be rewritten to give better time complexity bounds. The main result of this section however, is the following.

THEOREM 5.2. *A minimum height elimination tree for an interval graph can be found in polynomial time.*

It turns out that interval graphs are among the most useful mathematical structures for modeling real world problems. Applications of interval graphs can be found in several areas like mathematics, chemistry, biology and archeology. In the final section we will see the role of interval graphs in sparse factorization. We will also look at the place of $\Omega$ among the subclasses of interval graphs. Golumbic gives a detailed study of

```
Algorithm Interval (T: Clique tree; Opt: Array of integers);
begin
  for i = 1 to m do
    for j = 1 to m do
      Opt [i,j] = ∞;
  for i = m downto 1 do
    for j = i to m do
      A = {v ∈ C_k | k < i or k > j};
      for k = i to j − 1 do
        C'_k = C_k − A;
        S'_k = S_k − A;
      end-for;
      C'_j = C_j − A;
      if i = j then
        if C'_i is empty then
          Opt [i,j] = 0
        else
          Opt [i,j] = |C'_i| − 1;
      else
        last = i;
        while (S'_last ≠ ∅) and (last < j) do
          last = last + 1;
        if last = j then  (* I_ij is connected *)
          for k = i to j − 1 do  (* Pick S'_k as the top separator *)
            for l = i to j do
              C"_l = C'_l − S'_k;
              S"_l = S'_l − S'_k;
            start = i; stop = i; max = 0;
            while stop ≤ j do  (* Check minimum height for each component *)
              while (S"_stop ≠ ∅) and (stop < j) do
                stop = stop + 1;
              if max < Opt [start,stop] then
                max = Opt [start,stop];
              start = stop + 1;
              stop = stop +1;
            end-while;
            min = |S'_k| + max;
            if min < Opt [i,j] then
              Opt [i,j] = min;
          end-for;
        end-if;
      end-if;
    end-for;
end;
```

Figure 5.1: Algorithm *Interval*.

interval graphs in [13].

## 6 Concluding remarks

The main contribution of this paper has been to introduce a polynomial-time algorithm for finding minimum height elimination trees for interval graphs. In this last section, we give a couple of interesting results on interval graphs and the class $\Omega$ that might help to understand how widely applicable these graphs are. We start by looking at the matrices that correspond to interval graphs and see what interval graph property means for Cholesky factorization of such matrices.

### 6.1 Dense profile graphs

Let $G = (V, E) = G(A)$ for a symmetric, positive definite matrix $A$. One method that aims to reduce the fill-in size in Cholesky factorization of $A$ is the *bandwidth method.* For each vertex $v_i$ in $V$, let $b(v_i) = \max\{|i - j| \mid (v_i, v_j) \in E\}$. The *bandwidth* of $G$ is defined as $\max\{b(v_i) \mid v_i \in V\}$. The bandwidth method finds a band of width $2b + 1$ about the main diagonal of $A$ outside of which all the entries are zero, and performs Cholesky factorization within the band.

The problem of finding an ordering of the vertices in $G$ that results in minimum bandwidth for $G$ is NP-hard. Various heuristics exist for finding orderings with small bandwidth. Cuthill and McKee give an algorithm in [6] for this purpose.

An extension of the bandwidth method is the *profile method.* The *envelope* of $G$ is $\{(v_i, v_j) \mid i \leq j < b(v_i)\}$. The *profile* of $G$ is $|envelope(G)|$. The profile method assumes that all the non-zero entries in $A$ are within an envelope of varying width about the main diagonal. As with the bandwidth method, the problem of finding an ordering that results in minimum profile for $G$ is NP-hard.

A graph $G$ for which there is an ordering such that all edges within the envelope are present is defined by Tarjan in [24] as a *dense profile graph.* That is, $G$ is a dense profile graph if and only if there is an ordering of the vertices in $G$ such that if $(v_i, v_j) \in E$ and $i \leq k \leq j$ then $(v_k, v_j) \in E$. The following interesting result is from [24].

THEOREM 6.1. *$G$ is a dense profile graph if and only if $G$ is an interval graph.*

This result helps us to visualize what the corresponding matrices of interval graphs look like. An example is given in Figure 6.1. Also, any graph $G$ can be ordered so that the associated graph $G'$ of the envelope of $G$ is an interval graph. Then Algorithm *Interval* can be used to find a minimum height elimination tree for $G'$. Of course, this will not necessarily be a minimum height elimination tree for $G$, but if the profile of $G$ is small, then the elimination tree found will probably be close to the optimal one for $G$.

We now continue by looking at a known subclass of interval graphs called proper interval graphs and examining the relationship between this subclass and $\Omega$.

### 6.2 Proper interval graphs

The following theorem from [22] defines proper interval graphs.

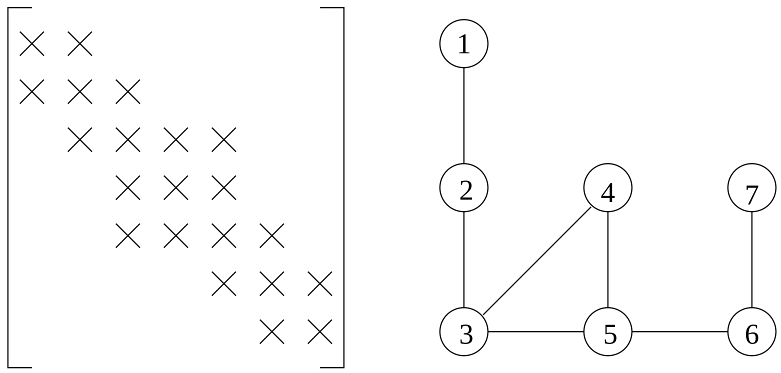


Figure 6.1: A dense profile matrix and its associated graph.

THEOREM 6.2. *$G$ is a proper interval graph if and only if $G$ is an interval graph containing no induced copy of $K_{1,3}$.*

The complete bipartite graph $K_{1,3}$ is shown in Figure 6.2 a). We will now show that the class $\Omega$ is not comparable to the class of proper interval graphs.

THEOREM 6.3. *There exist proper interval graphs that are not in $\Omega$, and there exist graphs in $\Omega$ that are not proper interval graphs.*

PROOF. Let $G$ be as shown in Figure 6.2 b). Clearly, $G \in \Omega$. But the black vertices in $G$ induce $K_{1,3}$, so $G$ cannot be a proper interval graph. Thus $\Omega \not\subseteq$ proper interval graphs. Let now $H$ be as shown in Figure 6.2 c). Since $H$ is an interval graph and does not contain an induced copy of $K_{1,3}$, $H$ is a proper interval graph. But $H$ is not in $\Omega$ since $H$ has more than two maximal cliques with simplicial vertices. Thus proper interval graphs $\not\subseteq \Omega$. □

Theorem 6.3 shows that $\Omega$ defines a subclass of interval graphs that is neither a subset nor a superset of proper interval graphs. This is an interesting result since it shows that $\Omega$ is a non-trivial subclass of interval graphs.

## 6.3 Open problems

Chordal graphs are a very important class of graphs in sparse Cholesky factorization. We have seen that all filled graphs are chordal and that all chordal graphs have perfect elimination orderings. It would be very interesting to know whether minimum height elimination trees of chordal graphs can be obtained in polynomial time.

It would also be interesting to know how much fill-in is introduced when graphs are reordered according to the algorithms that we have introduced. It has been conjectured that a minimum height elimination tree for a graph $G$ does not produce more than linear amount of extra fill-in compared to minimum fill-in size for $G$. Manne shows this when $G$ is a tree in [19]. We have not been able to find a linear bound for the amount of fill-in for any of the classes that appear in this paper. Even the class $\Omega$, which is the most restricted, seems to possess many of the properties of general graphs that make

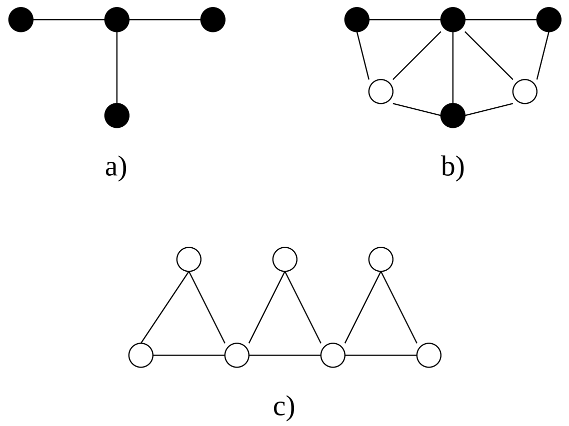


Figure 6.2: a) $K_{1,3}$ b) $G$ in the proof of Theorem 6.3. c) $H$ in the proof of Theorem 6.3.

it difficult to prove a linear bound.

**Postscript**

This paper is an extension of some of the results presented in [15]. The authors would like to thank Barry Peyton, and the referees, for many valuable suggestions.

For the minimum height elimination tree problem, the class of graphs with known polynomial time algorithms has recently been extended even further. In the context of vertex ranking, Deogun, Kloks, Kratsch and Müller [7] have presented an $O(n^6)$ algorithm for permutation graphs.

## REFERENCES

1. P. A. Bernstein and N. Goodman, *Power of natural semijoins*, SIAM J. Comput., 10 (1981), pp. 751–771.
2. J. R. S. Blair and B. W. Peyton, *An introduction to chordal graphs and clique trees*, pp. 1–30. In George, Gilbert and Liu [11].
3. J. R. S. Blair and B. W. Peyton, *On finding minimum-diameter clique trees*, Tech. Report ORNL/TM-11850, Oak Ridge National Laboratory, Tennessee, 1991.
4. K. S. Booth and G. S. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 335–379.
5. J. R. Bunch and D. J. Rose, eds., *Sparse Matrix Computations*, Academic Press, 1976.
6. E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, Proceedings of the $24^{th}$ National Conference of the ACM, (1969), pp. 157–172.
7. J. S. Deogun, T. Kloks, D. Kratsch, and H. Müller, *On vertex ranking for permutation and other graphs*, Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science, 775 (1994), pp. 747–758.

8. G. A. Dirac, *On rigid circuit graphs*, Abh. Math. Sem. Univ. Hamburg, 25 (1961), pp. 71–76.

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

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

11. J. A. George, J. R. Gilbert, and J. W. H. Liu, eds., *Graph Theory and Sparse Matrix Computations*, IMA Volumes in Mathematics and its Applications, Springer Verlag, 1993.

12. P. C. Gilmore and A. J. Hoffman, *A characterization of comparability graphs and of interval graphs*, Canadian Journal of Mathematics, 16 (1964), pp. 539–548.

13. M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, 1980.

14. F. Harary, ed., *Proof Techniques in Graph Theory*, Academic Press, 1969.

15. P. Heggernes, *Minimizing fill-in size and elimination tree height in parallel Cholesky factorization*, master's thesis, University of Bergen, Norway, 1992.

16. C. W. Ho and R. C. T. Lee, *Counting clique trees and computing perfect elimination schemes in parallel*, Information Processing Letters, 31 (1989), pp. 61–68.

17. J. W. H. Liu, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134–172.

18. F. Manne, *Reducing the height of an elimination tree through local reorderings*, Tech. Report 51, University of Bergen, Norway, 1991.

19. F. Manne, *An algorithm for computing a minimum height elimination tree for a tree*, Tech. Report 59, University of Bergen, Norway, 1992.

20. A. Pothen, *The complexity of optimal elimination trees*, Tech. Report CS-88-13, Pennsylvania State University, 1988.

21. R. E. Read, ed., *Graph Theory and Computing*, Academic Press, 1972.

22. F. S. Roberts, *Indifference graphs*, pp. 139–146. In Harary [14].

23. D. J. Rose, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, pp. 183–217. In Read [21].

24. R. E. Tarjan, *Graph theory and Gaussian elimination*, pp. 3–22. In Bunch and Rose [5].

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. Computing, 8 (1984), pp. 566–579.

26. M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Alg. Disc. Meth., 2 (1981), pp. 77–79.