A Vertex Incremental Approach for Maintaining Chordality

Anne Berry* Pinar Heggernes† Yngve Villanger†

Abstract

For a chordal graph $G = (V, E)$, we study the problem of whether a new vertex $u \notin V$ and a given set of edges between u and vertices in V can be added to G so that the resulting graph remains chordal. We show how to resolve this efficiently, and at the same time, if the answer is no, specify a maximal subset of the proposed edges that can be added along with u, or conversely, a minimal set of extra edges that can be added in addition to the given set, so that the resulting graph is chordal. In order to do this, we give a new characterization of chordal graphs and, for each potential new edge uv, a characterization of the set of edges incident to u that also must be added to G along with uv. We propose a data structure that can compute and add each such set in $O(n)$ time. Based on these results, we present an algorithm that computes both a minimal triangulation and a maximal chordal subgraph of an arbitrary input graph in $O(nm)$ time, using a totally new vertex incremental approach. In contrast to previous algorithms, our process is on-line in that each new vertex is added without reconsidering any choice made at previous steps, and without requiring any knowledge of the vertices that might be added subsequently.

1 Introduction

Chordal graphs (also called triangulated graphs) are a well-studied class of graphs, with applications in many fields. Some applications requires that chordality be maintained incrementally, that is, as edges and/or vertices are added or deleted from the graph, they desire to maintain chordality. Ibarra [27] gives a dynamic algorithm for adding or removing a given edge in $O(n)$ time in a chordal graph if this does not destroy chordality, where n is the number of vertices of the input graph. More recently, a *2-pair* [10] has been defined as pair of non adjacent vertices in a chordal graph, such that the graph remains chordal when the edge between these vertices is added to the graph.

A chordal graph can be obtained from any non chordal graph by: adding edges until the graph becomes chordal, a process called *triangulation*, or by removing edges until the graph becomes chordal, thus computing a *chordal subgraph*. Adding or removing a minimum number of edges has been shown to be NP-hard [29, 35]. However, adding or removing an inclusion minimal set of edges can be done in polynomial time. Given an arbitrary chordal subgraph (e.g., an independent set on the vertices of the graph) or supergraph (e.g., a complete graph on the same vertex set) of the input graph, edges can be added or removed one by one after testing that the resulting graph remains chordal, until no further candidate edge can be found. This ensures that minimality is achieved, by the results of [31]. The problem of maintaining a chordal graph by edge addition or deletion and the problem of computing a maximal chordal subgraph or a minimal chordal supergraph are thus strongly related.

The problem of adding an inclusion minimal set of *fill* edges, called *minimal triangulation*, has many applications in various fields such as sparse matrix computation [30] and database

*LIMOS, UMR CNRS 6158, Universite Clermont-Ferrand II, F-63177 Aubiere, France. berry@isima.fr

†Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Pinar.Heggernes@ii.uib.no yngvev@ii.uib.no

management [3]. The problem has been well studied since 1976, and several $O(nm)$ time algorithms exist for solving it [4, 5, 8, 15, 31], where m is the number of edges in the input graph. None of these algorithms use an edge incremental approach as described above. However, the algorithm proposed by Blair, Heggernes, and Telle [11], which requires even less time when the fill is small, does use an edge deletion approach.

The reverse problem of computing a maximal chordal subgraph has also been studied, with applications to sparse matrix computation, computing a large clique or a large independent set, and improving phylogenetic data [2, 10, 14, 16, 19, 34]. There exist several algorithms that compute a maximal chordal subgraph in $O(\Delta m)$ time, where Δ is the maximum degree in the graph [2, 16, 34].

In this paper, we present a new process for adding a *vertex* with a given set of incident edges to a chordal graph while maintaining chordality, which we are able to implement more efficiently than if we were to add the corresponding edges one by one. Our process is based on two new characterizations. The first is a characterization of a chordal graph by its edges, which can be regarded as a specialization of the edge characterization for weakly chordal graphs introduced by Berry, Bordat, and Heggernes [7]. The second is a characterization of a unique set of edges $R(G, u, v)$ incident to a vertex u that must be added to a chordal graph G along with edge uv to ensure that chordality is preserved, given that we are only allowed to add edges incident to u. We show that we can compute this set $R(G, u, v)$ of edges in $O(n)$ time, by proposing a data structure that corresponds to a clique tree of the current chordal subgraph. A similar data structure was used by the authors to prove an $O(nm)$ time bound for one of their minimal triangulation algorithms [4, 8, 25]; however, here we present a new implementation of clique trees that allows a more efficient data structure for our purposes.

We use our results to compute both a minimal triangulation and a maximal chordal subgraph of a given arbitrary graph in $O(nm)$ time. This is done by an incremental process that repeatedly adds a new vertex u to the already constructed chordal graph H along with a maximal set of edges between u and H, or a minimal set of extra edges between u and H in addition to the originally specified edges.

Some of the existing algorithms that compute a maximal chordal subgraph or a minimal triangulation also use a vertex incremental process [2, 5, 8, 16, 31, 34], though none of them compute both chordal graphs at the same time. In addition, all these previous algorithms require knowing the whole graph in advance, as either vertices that are not yet processed are marked in some way to define the next vertex in the process, or edges are added between pairs of vertices that are not yet processed. Furthermore, these algorithms require the added vertex to be a simplicial vertex of the transitory chordal graph. One exception from this requirement is the algorithm of [8], but it does add edges between pairs of neighbors of the added vertex that are not yet processed.

The approach here is completely different from the previous ones, as it is more general: at each vertex addition step, we do not require the added vertex to be or to become simplicial, thereby enabling processing of vertices in any order. Moreover, we add only edges incident to the new vertex, so that we never need to reconsider or change the chordal graph that has been computed thus far.

As a result, our process can add any vertex with any proposed neighborhood, and efficiently give a correction if the resulting graph fails to be chordal, either by computing a maximal subset of the edges to be added, or a minimal set of extra edges along with the proposed ones. In addition, the transitory chordal graph is maintained in a dynamic fashion, as making the desired or necessary additions to the graph does not require a recomputation.

This paper is organized as follows: in the next section we give the necessary graph theoretic background and terminology. Section 3 contains our new characterizations. The algorithms are presented and proved correct in Section 4, whereas the data structure details and time complexity analysis are given in Section 5. We conclude in Section 6.

2 Graph theoretic background and notation

We assume that all graphs are simple and connected. For disconnected graphs, our algorithm can be applied on each connected component. A graph is denoted $G = (V, E)$, with $n = |V|$, and $m = |E|$. A vertex sequence $v_1 - v_2 - ... - v_k$ describes a *path* if $v_i v_{i+1}$ is an edge for $1 \leq i < k$. The *length* of a path is the number of edges in it. A *cycle* is a path that starts and ends with the same vertex, and the length of the cycle is the number of vertices or edges it contains. A *chord* of a cycle (path) is an edge connecting two non-consecutive vertices of the cycle (path). A *clique* is a set of vertices that are pairwise adjacent. A *simplicial* vertex is one whose neighborhood induces a clique.

For the following definitions, we will omit subscript G when the graph is clear from the context. The *neighborhood* of a vertex v in G is $N_G(v) = \{u \neq v \mid uv \in E\}$, and for a set of vertices A, $N_G(A) = \cup_{x \in A} N_G(x) - A$. $G(A)$ is the subgraph induced by a vertex set $A \subseteq V$, but we often denote it simply by A when there is no ambiguity. We would like to stress that we distinguish between subgraphs and induced subgraphs. When we say merely *subgraph* we mean a proper subgraph on the same vertex set with fewer edges.

For any vertex set $S \subseteq V$ and any vertex $x \in V \setminus S$, C_S^x denotes the connected component of $G(V \setminus S)$ containing x. A subset S of V is called a *separator* if $G(V \setminus S)$ is disconnected. S is a *u, v-separator* if vertices u and v are in different connected components of $G(V \setminus S)$, and a *minimal u, v-separator* if no subset of S is a u, v-separator. S is a *minimal separator* of G if there is some pair $\{u, v\}$ of vertices in G such that S is a minimal u, v-separator. Equivalently, S is a minimal separator if there exist two connected components C_1 and C_2 of $G(V \setminus S)$ such that $N_G(C_1) = N_G(C_2) = S$.

A pair of non-adjacent vertices $\{u, v\}$ is a *2-pair* in G if there is no chordless path of length 3 or more between u and v [23]. If G is not connected, then two vertices that belong to different connected components constitute a 2-pair by definition. If G is connected, it has been shown that $\{u, v\}$ is a 2-pair if and only if $N(u) \cap N(v)$ is a minimal u, v-separator of G [1, 32].

A graph is *chordal* if it contains no chordless cycle of length ≥ 4. Consequently, all induced subgraphs of a chordal graph are also chordal. G is chordal if and only if every minimal separator of G is a clique [18]. Chordal graphs are the intersection graphs of subtrees of a tree [13, 21, 33], and the following result gives a very useful tool which we will use as a data structure in our algorithm.

Theorem 2.1 (Buneman [13], Gavril [21], Walter[33]) *A graph G is chordal if and only if there exists a tree T, whose vertex set is the set of maximal cliques of G, that satisfies the following property: for every vertex v in G, the set of maximal cliques containing v induces a connected subtree of T.*

Such a tree is called a *clique tree* [12], and we will refer to the vertices of T as *tree nodes* to distinguish them from the vertices of G, and sometimes also as *bags* since these contain several graph vertices. Each tree node of T is thus a vertex set of G corresponding to a maximal clique of G. We will not distinguish between cliques of G and their corresponding tree nodes. In addition, it is customary to let each edge $K_i K_j$ of T hold the vertices of $K_i \cap K_j$, where K_i and K_j are maximal cliques of G. Thus, edges of T are also vertex sets. Although a chordal graph can have many different clique trees, all chordal graphs share the following important properties that are related to an efficient implementation of our algorithm.

Theorem 2.2 (Buneman [13], Ho and Lee[26], Lundquist [28]) *Let T be a clique tree of a chordal graph G. A set S is a minimal separator of G if and only if $S = K_i \cap K_j$ for an edge $K_i K_j$ in T, and if $S = K_i \cap K_j$ for an edge $K_i K_j$ in T, then S is a minimal u, v-separator for any $u \in K_i \setminus S$ and $v \in K_j \setminus S$.*

Theorem 2.3 (Blair and Peyton [12]) *T is a clique tree of G if and only if for every pair of distinct maximal cliques K_i and K_j in G, the intersection $K_i \cap K_j$ is contained in every node of T (maximal clique of G) appearing on the path between K_i and K_j in T.*

Note that as a consequence, the intersection $K_i \cap K_j$ is also contained in every edge of T (i.e., in every minimal separator of G) appearing on the path between K_i and K_j in T. A chordal graph has at most n maximal cliques [18] and hence the number of nodes and edges in a clique tree is $O(n)$ [20].

From any given non-chordal graph, one can obtain a chordal graph on the same vertex set by either adding (the added edges are called *fill* edges) or removing edges. $M = (V, F)$ is called a *triangulation* of an arbitrary graph $G = (V, E)$ if $E \subseteq F$ and M is chordal. M is a *minimal triangulation* of G if no proper subgraph of M is a triangulation of G. Similarly, $H = (V, D)$ is called a *chordal subgraph*, or equivalently a *subtriangulation*, of G if $D \subseteq E$ and H is chordal. H is a *maximal chordal subgraph*, or a *maximal subtriangulation*, if (V, D') is non-chordal for every set D' that satisfies $D \subset D' \subseteq E$. By the results of [31], a given triangulation (subtriangulation) is minimal (maximal) if and only if no single fill edge can be removed (no single removed edge can be added back) without destroying chordality.

3 A new characterization of chordal graphs

In this section we present a new characterization of chordal graphs that will be the basis of our algorithm.

Definition 3.1 *An edge uv is* mono-saturating *in $G = (V, E)$ if $\{u, v\}$ is a 2-pair in $G' = (V,\ E \setminus \{uv\})$.*

Theorem 3.2 *A graph is chordal if and only if every edge is mono-saturating.*

Proof. Let $G = (V, E)$ be chordal, and assume on the contrary that there is an edge $uv \in E$ that is not mono-saturating. Then there is a chordless path of length more than 2 between u and v in G'. Thus, G' is connected, and $N(u) \cap N(v)$ is not a u, v-separator in G'. Let us remove $N(u) \cap N(v)$ from G'. There is still a path connecting u and v in the remaining graph. Let p be a shortest such path. Now, p contains a vertex $x \in N(u)$ that is not adjacent to v, and a vertex $z \in N(v)$ that is not adjacent to u. Thus, the following is a chordless cycle of length at least 4 in G: $u - p - v - u = u - x - ... - z - v - u$, which contradicts our assumption that G is chordal.

For the other direction, let every edge in G be mono-saturating, and assume on the contrary that G is not chordal. Thus, there exists a chordless cycle C of length at least 4 in G. Let uv be any edge of C. Since at least one vertex of C must be removed to separate u and v in G', any minimal u, v-separator of G' contains a vertex x of C, where $x \notin N(u)$ or $x \notin N(v)$. Therefore $N(u) \cap N(v)$ cannot be a u, v-separator, which contradicts our assumption that every edge is mono-saturating. ∎

As a corollary of Theorem 3.2, we can deduce the following characterization by 2-pairs by [10], a result that was also observed with a different formulation in [17].

Corollary 3.3 (Berry et al. [10]) *Given a chordal graph $G = (V, E)$, where $uv \notin E$, the graph $H = (V,\ E \cup \{uv\})$ is chordal if and only if $\{u, v\}$ is a 2-pair in G.*

Proof. Let us on the contrary assume that H is not chordal, and that $\{u, v\}$ is a 2-pair in G. The edge uv is mono-saturating in H since $\{u, v\}$ is a 2-pair of G. By Theorem 3.2 there exists an edge xy in G that is not mono-saturating in H, and by Definition 3.1 there exists an chordless path P in $H' = (V,\ (E \setminus \{xy\}) \cup \{uv\})$ preventing xy from being mono-saturated in

H. One of the edges in P is uv, since G is chordal and by Theorem 3.2 xy is mono-saturating in G, and by Definition 3.1 P do not exists in $G' = (V, E \setminus \{xy\})$. By removing the edge uv from P and inserting xy we obtain a chordless path P' in G, which prevents $\{u, v\}$ from being a 2-pair in G, and thus we have a contradiction.

For the other direction, we know that $\{u, v\}$ is not a 2-pair in G, and thus the edge uv in H is not mono-saturating, and by Theorem 3.2 H is not chordal. ∎

As a consequence, while maintaining a chordal graph by adding edges, we could check every edge of the input graph to see if the endpoints constitute a 2-pair in the transitory chordal subgraph. However, this approach requires that we check every edge several times, as pairs of vertices can become 2-pairs only after the addition of some other edges. Our main result, to be presented as Theorem 3.8, gives a more powerful tool that allows examining each edge of the input graph only *once* during such a process.

Assume the following scenario: given a chordal graph G, we want to add an edge uv to G. Since we want the resulting graph to remain chordal, it may be necessary to add other edges to achieve this. However, the addition of only edges incident to u. [1] Naturally, if we add every edge between u and the other vertices of G, the resulting graph is chordal. However, our main goal is to add as few edges as possible. Theorem 3.8 gives a necessary and sufficient condition for the addition of each such edge uv. Before we present it, we need the following definition for ease of notation.

Definition 3.4 *Given a chordal graph $G = (V, E)$ and any pair of non-adjacent vertices u and v in G, $R(G, u, v) = \{ux \mid x$ belongs to a minimal u, v-separator of $G\}$. We will call $R(G, u, v)$ the incident to u set of required edges for "uv".*

Lemma 3.5 *Let $G = (V, E)$ be a chordal graph and let u, v be non-adjacent vertices of G. $R(G, u, v) = \{ux \mid x$ is an intermediate vertex of a chordless path in G between u and $v\}$.*

Proof. Let $ux \in R(G, u, v)$, S be a minimal u, v-separator of G containing x, p_1 be a chordless path in G between u and x with all intermediate vertices belonging to C_S^u, and p_2 be a chordless path in G between x and v with all intermediate vertices belonging to C_S^v. The path obtained by concatenation of p_1 and p_2 is a chordless path in G between u and v having x as an intermediate vertex.
Conversely, let x be an intermediate vertex of a chordless path p in G between u and v. The vertex set S' obtained from V by removing all vertices of p except x is an uv-separator of G. Let S be a minimal u, v-separator of G included in S'. x belongs to S (otherwise p would be a path in $G(V \setminus S)$ between u and v), so $ux \in R(G, u, v)$. ∎

Lemma 3.6 *Let $G = (V, E)$ be a chordal graph, let u, v be non-adjacent vertices of G, let S be a minimal u, v-separator of G and let M be the graph $(V,\ E \cup \{uv\} \cup R(G, u, v))$. Any chordless cycle in M of length at least 4 containing u contains at most one vertex of S.*

Proof. Suppose on the contrary that some chordless cycle C in M of length at least 4 contains u and distinct vertices x and x' of S. As a minimal separator of a chordal graph, S is a clique in G, so xx' is an edge of M, and by definition of $R(G, u, v)$, ux and ux' are edges of M. So C has a chord in M, a contradiction. ∎

Lemma 3.7 *Let $G = (V, E)$ be a chordal graph, let u, v be non-adjacent vertices of G and let S, S' be minimal u, v-separators of G. Then ($S' \subseteq S \cup C_S^u$ and $S \subseteq S' \cup C_{S'}^v$) or ($S' \subseteq S \cup C_S^v$ and $S \subseteq S' \cup C_{S'}^u$).*

[1] Note that in the incremental approach described in the next section, vertex u is the most recently added vertex.

Proof. We may assume w.l.o.g. that S and S' are distinct. As minimal separators of a chordal graph, S and S' are cliques and since S and S' are distinct minimal u,v-separator of G. $S \setminus S' \neq \emptyset$. Let $x \in S \setminus S'$. $S' \cap (C_S^u \cup C_S^v) \neq \emptyset$ (otherwise $C_S^u \cup \{x\} \cup C_S^v$) would be a connected subset of $V \setminus S'$, which would contradict S' being an u,v-separator of G). Let $x' \in S' \cap (C_S^u \cup C_S^v)$.
First case : $x' \in C_S^u$
Since S' is a clique containing x', $S' \subseteq \{x'\} \cup N(x') \subseteq S \cup C_S^u$. It follows that $\{x\} \cup C_S^v)$ is a connected subset of $V \setminus S'$, so $x \in C_{S'}^v$. Since S is a clique containing x, $S \subseteq \{x\} \cup N(x) \subseteq S' \cup C_{S'}^v$.
Second case : $x' \in C_S^v$
We prove in a similar way that $S' \subseteq S \cup C_S^v$ and $S \subseteq S' \cup C_{S'}^u$. ∎

Theorem 3.8 *Let $G = (V, E)$ be a chordal graph, let u, v be non-adjacent vertices of G and let M be the graph $(V,\ E \cup \{uv\} \cup R(G,u,v))$. M is chordal and M is a subgraph of any triangulation of $G' = (V,\ E \cup \{uv\})$ obtained from G' by adding edges incident to u only.*

Proof. Let us show that M is chordal. Assume on the contrary that M is not chordal, and let C be a chordless cycle of length at least 4 in M. Since G is chordal C contains an edge $ux \in \{uv\} \cup R(G,u,v)$. Let $C = u - x' - y_1 - y_2 - ... - y_k - x - u$ with $k \geq 1$ and $p_1 = x' - y_1 - y_2 - ... - y_k - x$ which is a chordless path in G. It is sufficient to show that p_1 is a subpath of a chordless path p in G between u and v, since in that case by Lemma 3.5 uy_1 would belong to $R(G,u,v)$ and therefore would be a chord of C in M, a contradiction. In the following $q_1.q_2$ denotes the path obtained by concatenation of paths q_1 and q_2.
First case : $x = v$ or $x' = v$
Say, $x = v$. If $ux' \in E$ then we are done with $p = (u - x').p_1$. Otherwise $ux' \in R(G,u,v)$, let S be a minimal u,v-separator of G containing x' and let p_0 be a chordless path in G between u and x' with all intermediate vertices belonging to C_S^u. By Lemma 3.6, x' is the only vertex of S in p_1, so all intermediate vertices of p_1 belong to C_S^v. It follows that path $p = p_0.p_1$ is a chordless path in G between u and v.
Second case $x \neq v$ and $x' \neq v$
Then $ux \in R(G,u,v)$, let S be a minimal uv-separator of G containing x and let p_2 be a chordless path in G between x and v with all intermediate vertices belonging to C_S^v. If $ux' \in E$ then by Lemma 3.6, all intermediate vertices of $(u - x').p_1$ belong to C_S^u, so path $p = (u - x').p_1.p_2$ is a chordless path in G between u and v. Otherwise $ux' \in R(G,u,v)$, let S' be a minimal u,v-separator of G containing x' and let p_0 be a chordless path in G between u and x' with all intermediate vertices belonging to $C_{S'}^u$. By Lemma 3.7, ($S' \subseteq S \cup C_S^u$ and $S \subseteq S' \cup C_{S'}^v$) or ($S' \subseteq S \cup C_S^v$ and $S \subseteq S' \cup C_{S'}^u$). We may assume w.l.o.g. that $S' \subseteq S \cup C_S^u$ and $S \subseteq S' \cup C_{S'}^v$. By Lemma 3.6, $x' \notin S$ and $x \notin S'$, so $x' \in C_S^u$ and $x \in C_{S'}^v$. Hence by Lemma 3.6, all intermediate vertices of p_1 belong to $C_{S'}^v$ and C_S^u. Since p_0 and p_1 are chordless and all vertices of p_0 other than x' belong to $C_{S'}^u$ and those of p_1 other than x' belong to $C_{S'}^v$, path $q = p_0.p_1$ is a chordless. Since $S \subseteq S' \cup C_{S'}^v$, $C_{S'}^u \subseteq C_S^u$, so all vertices of p_0 belong to C_S^u. Since q and p_2 are chordless and all vertices of q other than x belong to C_S^u and those of p_2 other than x belong to C_S^v, path $p = q.p_2$ is a chordless path in G between u and v, which completes the proof of chordality of M.
Let M' be a triangulation of $G' = (V,\ E \cup \{uv\})$ obtained from G' by adding edges incident to u only. Let us show that M is a subgraph of M', i.e. that every edge of $R(G,u,v)$ is an edge of M'. Suppose on the contrary that there is some edge $ux \in R(G,u,v)$ which is not an edge of M'. By Lemma 3.5, x is an intermediate vertex of a chordless path $u - y_1 - y_2 - ... - y_i = x - ... - y_k = v$ in G. Thus $C = u - y_1 - y_2 - ... - y_k = v - u$ is a cycle in M' such that path $y_1 - y_2 - ... - y_k = v$ is chordless in M'. Let r be the greatest integer smaller than i such that y_r is adjacent to u in M' and s be the smallest integer greater than i such that y_r is adjacent to u in M'. Then $u - y_r - y_{r+1} - ... - y_i = x - ... - y_{s-1} - y_s - u$ is a chordless cycle in M' of length at least 4, which contradicts the chordality of M'. ∎

The following corollary follows directly from Theorem 3.8.

Corollary 3.9 *Let $G = (V, E)$ be a chordal graph and let u and v be any pair of non-adjacent vertices in G. Then $M = (V,\ E \cup \{uv\} \cup R(G, u, v))$ is a minimal triangulation of $(V,\ E \cup \{uv\})$.*

A significant consequence of our main theorem is that we can efficiently determine $R(G, u, v)$ and add it to the current graph G to obtain a new chordal graph, as will be explained in Section 5. This involves maintaining the minimal separator structure of a chordal graph, a problem for which we have a new and efficient data structure associated with a clique tree, which we will describe in Section 5.

4 A vertex incremental algorithm for simultaneous maximal subtriangulation and minimal triangulation

In this section we apply our results of Section 3 to the problem of computing a maximal chordal subgraph $H = (V, D)$ and a minimal triangulation $M = (V, F)$ of an arbitrary graph $G = (V, E)$, where $D \subseteq E \subseteq F$.

Our algorithm is based on the following vertex incremental principle. Start with an empty subset U of V, and a maximal chordal subgraph H of $G(U)$ (respectively a minimal triangulation M of $G(U)$ if we want a minimal triangulation algorithm). The incremental approach is to increase U with a vertex u from $V \setminus U$ at each step. Observe that H (resp. M) is chordal and disconnected after the introduction of u as long as $|U| > 1$, since no edges are introduced along with the vertex, and H (resp. M) was chordal before this step.

Then for each edge of G incident to u and some vertex v in $U \setminus \{u\}$, we do computations according to Theorem 3.8 and obtain the set $R(H, u, v)$ (resp. $R(M, u, v)$) of edges incident to u that must be added along with uv in order to obtain a chordal supergraph of H (resp. M).

In the case of the maximal subtriangulation algorithm, we will only add uv and $R(H, u, v)$ to $E(H)$ if $R(H, u, v) \subset E(G)$. In the case of the minimal triangulation algorithm, the required edges $R(M, u, v)$ and the edge uv are added to $E(M)$. To prove that this approach actually produces a maximal chordal subgraph (resp. minimal triangulation) we relay on the results in the following two lemmas.

Lemma 4.1 *Given $G = (V, E)$, let $H = (U, D)$ be a maximal chordal subgraph of $(U, E') = G(U)$, where $U \subset V$ and $D \subseteq E' \subseteq E$. No edge belonging to $E' \setminus D$ can be contained in a maximal chordal subgraph of G that also has H as a subgraph.*

Proof. Let uv be any edge of $E' \setminus D$. Thus, u and v both belong to the chordal subgraph H. Let $H' = (V, D')$ be a maximal chordal subgraph of G with $D \subset D'$, and assume on the contrary that uv belongs to D'. Since induced subgraphs of chordal graphs are also chordal, $H'(U)$ is chordal and contains edge uv. But this contradicts the assumption that H is a maximal chordal subgraph of $G(U)$, since $H'(U)$ is a chordal subgraph of $G(U)$ that contains H as a proper subgraph. ∎

For the computation of H, assume that H is a maximal chordal subgraph of $G(U)$ on vertex set U. At the first step, U contains a single vertex of G and $H = G(U)$. At each step, a vertex $u \in V \setminus U$ is chosen and added to U and thus to H. Now for each edge uv of G with $v \in U$, we add edge uv to H if and only if every edge of $R(H, u, v)$ is present in G. If uv is added to H, we also add every edge of $R(H, u, v)$ at the same time. After this, none of the edges that are added need to be examined again for possible addition, since they already appear in the transitory chordal subgraph. If some edge of $R(H, u, v)$ is not an edge of G, then we cannot add uv at this step by Theorem 3.8, and uv never needs to be examined again for addition by Lemma 4.1. Note that $R(H, v, u)$ does not help, since we have already concluded that adding

any edge between v and vertices in $U \setminus \{u\}$ will create a chordless cycle. Thus, each edge is examined for addition at most once, and in many cases several edges are added at the same time and disappear from the list of edges that still need to be examined, which is the strength of our algorithm with respect to time complexity. In addition, our algorithm does not touch the unprocessed vertices. Thus, these vertices need not be known in advance, and we can actually take a new vertex u as input in an on-line fashion at each step.

Lemma 4.2 *Given $G = (V, E)$, let $M = (U, F)$ be a minimal triangulation of $G(U)$ with $U \subset V$. Then any minimal triangulation of $(V, E \cup F)$ obtained by introducing only edges with at least one endpoint in $V \setminus U$ is a minimal triangulation of G.*

Proof. Let $M' = (V, F')$ be a minimal triangulation of $(V, E \cup F)$ obtained by introducing only edges with at least one endpoint in $V \setminus U$. M' exists by Theorem 3.8, and M' is certainly a triangulation of G since it is chordal and contains all edges of G. Assume on the contrary that M' is not a minimal triangulation of G. Thus, there is at least one edge in $F' \setminus E$ that can be removed. If this edge belongs to $F' \setminus (E \cup F)$, then this contradicts our assumption that M' is a minimal triangulation of $(V, E \cup F)$. Thus, an edge uv belonging to $F \setminus E$ can be removed from M' without destroying its chordality. However, since M is a minimal triangulation of $G(U)$, removing uv creates a chordless cycle C of length at least 4 in M. Since no edge of $F' \setminus F$ has both its endpoints in U, $F' \setminus F$ does not contain a chord of C, and consequently the vertices belonging to C will induce a chordless cycle in M' if uv is removed, giving the desired contradiction. ∎

For the computation of M, assume that M is a minimal triangulation of $G(U)$ on the vertex set U. The only difference from the discussion above in this case is that, for each edge uv of G with $v \in U$, we add to M edge uv as well as *every edge belonging to* $R(M, u, v)$ regardless of whether or not these edges belong to G. Thus, the difference between the two processes is merely a single **if** line. Our algorithm can be changed by inserting or deleting this **if** line in order to change between the processes of computing a minimal triangulation and a maximal chordal subgraph, though of course both graphs can be computed by a single algorithm within the same time bound.

With the data structure details given in the next section, we will show that computing and adding the set $R(H, u, v)$ can be done in $O(n)$ time for each examined edge uv. By Lemma 4.1 and 4.2 every edge needs to be examined at most once. We are now ready to present our algorithm. We begin with the maximal chordal subgraph version.

Algorithm Incremental Maximal Subtriangulation **(IMS)**
Input: $G = (V, E)$.
Output: A maximal chordal subgraph $H = (V, D)$ of G.
Pick a vertex s of G;
$U = \{s\}$;
$D = \emptyset$;
for $i = 2$ to n **do**
    Pick a vertex $u \in V \setminus U$;
    $U = U \cup \{u\}$;
    $N = \emptyset$;
    for each vertex $w \in N_G(u)$
        if $w \in U$ **then**
            $N = N \cup \{w\}$;
    while N is not empty **do**
        Pick a vertex $v \in N$;
        $N = N \setminus \{v\}$;
        $X = \{x \mid x$ belongs to a minimal u, v-separator of $H = (U, D)\}$;

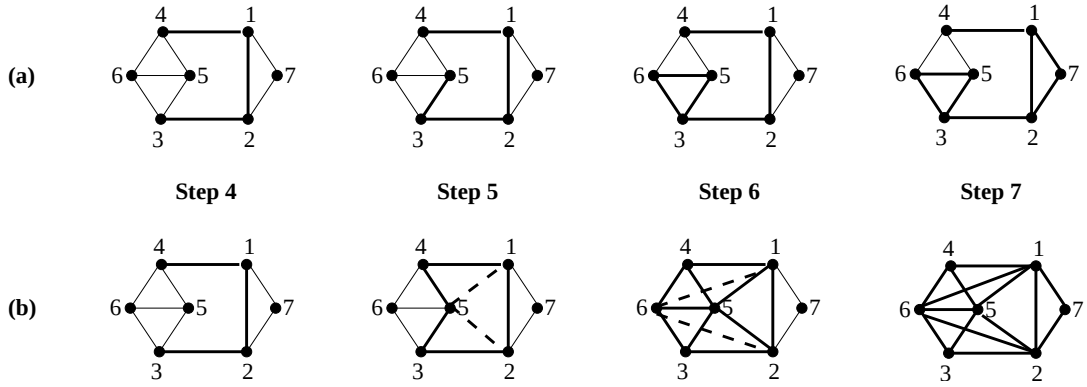

Figure 1: The figure shows graph H in thick lines after steps 4, 5, 6, and 7 of (a) Algorithm **IMS** when computing a maximal chordal subgraph and (b) Algorithm **IMT** when computing a minimal triangulation.

$R = \{ux \mid x \in X\}$;
if $R \subseteq E$ **then**
$D = D \cup \{uv\} \cup R$;
$N = N \setminus X$;
$H = (U, D)$;

Let us call **IMT** (Incremental Minimal Triangulation) the algorithm that results from removing line "**if** $R \subseteq E$ **then**" of Algorithm **IMS**. Thus, in **IMT**, edge set $\{uv\} \cup R$ is always added to the transitory graph for every examined edge uv. In Example 4.3, executions of both of these algorithms are shown on the same input graph. Figure 1 (a) shows **IMS** and (b) shows **IMT**.

Example 4.3 *Consider Figure 1. The vertices of the input graph are processed in the order shown by the numbers on the vertices. At step 1, only vertex 1 is added to H. At step 2, vertex 2 and edge* 21 *are added, and similarly at steps 3 and 4, vertex 3 and edge* 32*, and vertex 4 and edge* 41 *are added, respectively. The first column of the figure shows graph H with thick lines on the input graph after these 4 steps. Graph H so far is the same for both a maximal chordal subgraph (a), and a minimal triangulation (b). We will explain the rest of the executions in more detail.*

(a) *At step 5, $N = \{3, 4\}$, and edge* 53 *is examined first. In this case, set X is empty, and edge* 53 *is thus added. For the addition of edge* 54*, $X = \{1, 2, 3\}$, and since required edges* 51 *and* 52 *are not present in G, edge* 54 *is not added. At step 6, $N = \{3, 4, 5\}$, and edge* 63 *is examined first and added since X is empty. For the addition of edge* 64*, $X = \{1, 2, 3\}$, and since required edges* 61 *and* 62 *are not present in G, edge* 64 *is not added. For the addition of edge* 65*, $X = \{3\}$, and* 65 *is added since edge* 63 *is present in G and in H.*

(b) *At step 5, edge* 53 *is added as in (a), and in addition, edge* 54 *is added along with the required edges* 51 *and* 52*. At step 6, edge* 63 *is added as in (a). For the addition of edge* 64*, $X = \{1, 2, 3, 5\}$ since the minimal $6, 4$-separators are $\{1, 5\}, \{2, 5\}$, and $\{3\}$. Thus, edge* 64 *and required edges* 61*,* 62*, and* 65 *are added to H.*

Step 7 adds edges 71 *and* 72 *in both (a) and (b) without requiring any additional edges in either case.*

The following theorem follows by straight forward induction from Theorem 3.8 and Lemmas 4.1 and 4.2.

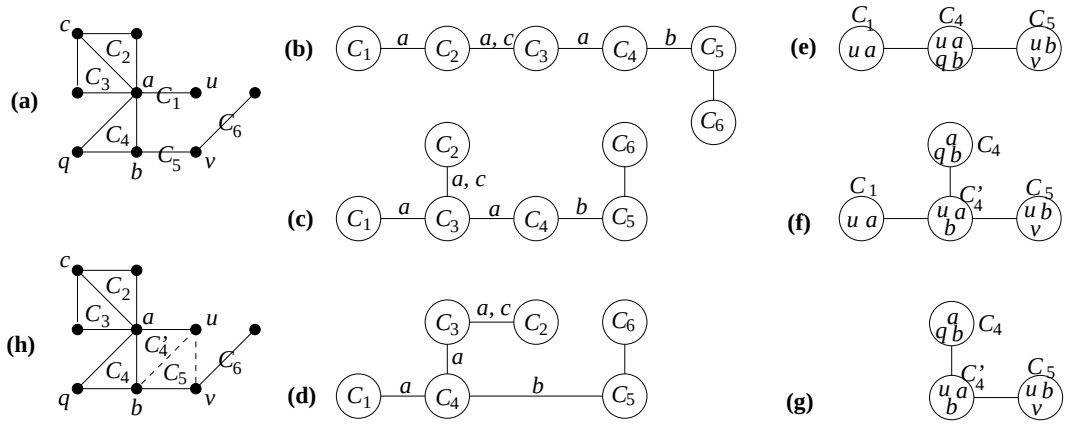

Figure 2: A chordal graph H is given in (a), and (b),(c), and (d) shows a clique tree of H, where $C_u = C_1$, $C_v = C_5$, and $P_{u,v}$ is the path between C_1 and C_5. After steps (c) and (d), path $P_{u,v}$ between C_1 and C_5 is in the desired form, and only this portion of the tree is shown after step (d). In step (e), u is placed in every clique on $P_{u,v}$, and in step (f) C_4 is separated from the path since edge uq is not intended. C_1 is removed in (g) since it becomes non-maximal. The new corresponding graph H of which the modified tree is a clique tree is shown in (h).

Theorem 4.4 *Algorithm* **IMS** *computes a maximal chordal subgraph, and Algorithm* **IMT** *computes a minimal triangulation of the input graph.*

5 Data structure and time complexity

The input graph G is represented by adjacency list data structure, and we use a clique tree T of H as an additional data structure to store and work on the transitory graph H. Thus, after the first step, T has only one tree node, which contains start vertex s. As H grows, T will grow maintaining a correct clique tree of H at all steps. Note that T will not always be connected at intermediate steps. We could have picked the new vertex u such that u is contained in the neighborhood of some vertex already contained in U, but this will be less general.

In what follows we describe an implementation of each of the following operations.

1. Compute the union X of all minimal u, v-separators in H, which gives the required edge set $R(H, u, v)$.

2. If $R(H, u, v) \cup \{uv\}$ is to be added to H, update T to reflect this modification of H.

As will be shown, each of these operations will be shown to require only $O(n)$ time for each examined edge uv of G.

We will devote a subsection to each of the above mentioned operations. Subsection 5.1 describes how T is modified to obtain a path $P_{u,v}$ such that every tree edge on $P_{u,v}$ is a distinct minimal u, v-separator (Figure 2(b)-(d)), and how the union X of all these minimal separators is computed from $P_{u,v}$. Subsection 5.2 describes how T is further modified to reflect the addition of new edges to H (Figure 2(e)-(g)), and how to ensure that every tree node in T is a unique maximal clique of H after the modifications.

Since we examine each edge at most once, and there are m edges, the desired time bound will then follow. An illustration of what happens for each examined edge uv is summarized in Figure 2; we will refer to parts of this figure as we explain the details in the coming subsections. The main idea is to use a path $P_{u,v}$ of the current clique tree T between a clique C_u that

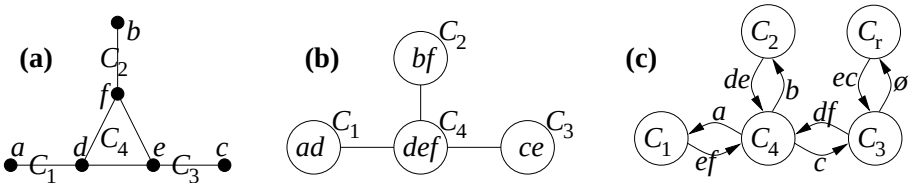

Figure 3: A chordal graph H and its maximal cliques are given in (a). A clique tree of H is given in (b), where each tree node contains the vertices of a unique maximal clique in H. The difference list representation of the same clique tree as in (b) is presented in (c). The clique C_r is an empty clique, that is used when we want to compute the set of vertices contained in a tree node. The arrows represents the add and remove lists, and the vertices contained in each list is given by the label attached to the arrow.

contains u and a clique C_v that contains v, and compute the union of tree edges on this path that correspond to minimal u, v-separators. Unfortunately, the sum of the sizes of these edges can be larger than $O(n)$; in fact each edge can be of size $O(n)$. Thus, if the tree nodes and tree edges of T are implemented simply as vertex lists containing vertices of each tree node and edge, then Operation 1 described above cannot be accomplished in $O(n)$ time. For this reason, we present a special kind of implementation of the clique tree, as described below.

Every edge CC' of T is implemented as two lists that we will call *difference lists* (*difflists* for short). One list contains vertices belonging to $C \setminus C'$. This list has two names; it is called both $add(C', C)$ and $remove(C, C')$. The other list contains vertices belonging to $C' \setminus C$. This list is called $add(C, C')$ and also $remove(C', C)$. Now, if every clique C of T contains pointers to its *add* and *remove* lists, then C actually does not need to store a list of vertices that it contains. Vertices belonging to C can be computed by using the edges incident to C as follows. For every edge CC', if we know the set of vertices belonging to C', then we add $add(C', C)$ to this set and remove $remove(C', C)$ from this set to get the set of vertices belonging to C. In order to have a starting clique we need to know the content of one clique, this might be an empty clique. Note that the add and remove lists and the structure of the tree is the only stored information in this data structure. In Figure 3 a chordal graph is given in (a), while the regular clique tree of this graph is given in (b), and the clique tree represented by difference lists is given in (c).

5.1 Finding the minimal u, v-separators and computing X

Let K_u be any maximal clique of H that contains u, and let K_v be any maximal clique that contains v. If K_u and K_v are contained in different connected components of T, then $X = \emptyset$ and there is nothing to compute. Let us for the rest of this subsection assume that K_u and K_v are contained in the same connected component of T.

On the path from K_u to K_v in T, do a search from K_u to K_v and let C_u be the clique closest to K_v containing u. Do a similar search from K_v to K_u and let C_v be We will let $P_{u,v}$ denote the path of T between C_u and C_v. An example graph H and $P_{u,v}$ are given in Figure 2 (a) and (b), respectively.

Claim 5.1 *Every minimal u, v-separator appears as an edge of $P_{u,v}$, and every edge on $P_{u,v}$ separates u and v.*

Proof. By Theorem 2.1 we know that $P_{u,v}$ exists. Theorem 2.2 ensures that every minimal u, v-separator S appears as an edge of $P_{u,v}$, since removing S will separate u and v in the graph, thus also separate C_u from C_v in any clique tree. Finally Theorem 2.3 ensures that every minimal separator S appearing as an edge of $P_{u,v}$ separates u and v, since S is either a minimal u, v-separator or contains a minimal u, v-separator as a subset. ■ However, some

of the minimal separators on $P_{u,v}$ might be non-minimal u,v-separators, and some minimal u,v-separators might appear several times on $P_{u,v}$. We will first modify T in $O(n)$ time so that path $P_{u,v}$ between C_u and C_v contains only distinct minimal u,v-separators as its edges.

Observe first that, since every vertex can appear only once in an *add* list and once in a *remove* list, the sum of the lengths of the *add* and *remove* lists on the path $P_{u,v}$ is at most $2n$. We obtain our time bound by reading every *add* and *remove* list in $P_{u,v}$ at most a constant number of times. The maximal cliques (tree nodes) on $P_{u,v}$ are named $C_1, C_2, ..., C_k$, where $C_u = C_1$ is the clique containing the vertex u, and $C_v = C_k$ is the clique containing v, and edge $S_i = C_iC_{i+1}$ is an edge of $P_{u,v}$, for $1 \leq i \leq k-1$. We will describe how unnecessary maximal cliques can be removed from $P_{u,v}$, and after each removal, we will assume that the remaining maximal cliques and minimal separators are resorted as $C_1, C_2, ...$, so that before we explain each new modification, we have a consecutive numbering of the maximal cliques on $P_{u,v}$. When we must remove edges of $P_{u,v}$ and insert edges between two non-consecutive maximal cliques of $P_{u,v}$, we will need a more general way of naming the intersections between two maximal cliques that are not necessarily adjacent in T. We let $S_{i,j} = C_i \cap C_j$ denote the intersection between C_i and C_j.

Claim 5.2 *Assume that there is a tree edge S_j on $P_{u,v}$, such that either S_j is not a minimal u,v-separator in H or S_j is equal to another minimal separator appearing on $P_{u,v}$. Then there exist two other consecutive (neighboring) tree edges S_i and S_{i+1} on $P_{u,v}$, such that $S_i \subseteq S_{i+1}$.*

Proof. Observe first that, by Theorems 2.1, 2.2, and 2.3, every minimal u,v-separator appears as an edge of $P_{u,v}$, and every edge on this path separates u from v. Thus, S_j is a (not necessarily minimal) u,v-separator. Consequently, there exists a minimal u,v-separator S_i such that $S_i \subseteq S_j$. Let S_{i+1} be the edge adjacent to S_i in $P_{u,v}$ in the direction of S_j. Note that $i+1$ and j might be equal. It follows from Theorem 2.3 that $S_i \subseteq S_{i+1}$ since $S_i \subseteq S_j$. ∎

From Claim 5.2, we can conclude that, if no two adjacent tree edges are subsets of or equal to each other, then $P_{u,v}$ contains only distinct minimal u,v-separators. We will test adjacent tree edges on $P_{u,v}$, and remove the ones that include their neighbors as a subset. In order to obtain our time bound we have to do this test in the difflist data structure.

Claim 5.3 *Let the following path C_i, S_i, C_j, S_j, C_l be a subpath of $P_{u,v}$. Then $S_i \subseteq S_j$ if and only if $remove(C_j, C_l) \subseteq add(C_i, C_j)$.*

Proof. Assume that $remove(C_j, C_l) \subseteq add(C_i, C_j)$. Observe that $S_i \cup add(C_i, C_j) = S_j \cup remove(C_j, C_l) = C_j$, since $S_i = C_i \cap C_j$ and $S_j = C_j \cap C_l$. Remember that *add* and *remove* lists only contain the new vertices, and thus $S_i \cap add(C_i, C_j) = S_j \cap remove(C_j, C_l) = \emptyset$. We can now conclude that $S_i \subseteq S_j$, since $remove(C_j, C_l) \subseteq add(C_i, C_j)$ and $S_i \cup add(C_i, C_j) = S_j \cup remove(C_j, C_l)$. For the other direction, assume that $S_i \subseteq S_j$. By the same arguments as in the opposite direction it follows that $remove(C_j, C_l) \subseteq add(C_i, C_j)$, since $S_i \cup add(C_i, C_j) = S_j \cup remove(C_j, C_l)$ and $S_i \cap add(C_i, C_j) = S_j \cap remove(C_j, C_l) = \emptyset$. ∎

Assume that $S_i \subseteq S_j$ in the subpath C_i, S_i, C_j, S_j, C_l of $P_{u,v}$. We update T to reduce $P_{u,v}$ as follows: we remove edge S_i from T, and we insert an edge $S_{i,l} = C_i \cap C_j$ between C_i and C_l. Thus, after this modification, the remaining subpath of $P_{u,v}$ between C_i and C_l is merely $C_i, S_{i,l}, C_l$. This situation corresponds to the change from (b) to (c) in Figure 2.

Claim 5.4 *Let the following path C_i, S_i, C_j, S_j, C_l be a subpath of $P_{u,v}$ such that $S_i \subseteq S_j$, then $S_i = S_{i,l}$.*

Proof. Let us assume to the contrary that $S_i \not\subseteq S_{i,l}$, then there exist a vertex $x \in S_i \setminus S_{i,l}$. Vertex x belongs to S_j, since $S_i \subseteq S_j$. Thus, $x \in C_l$ since $S_j = C_j \cap C_l$. This is a contradiction since $S_{i,l} = C_i \cap C_l$, and $x \notin S_{i,l}$, and x is contained in both C_i and C_l. For the opposite

direction, let us on the contrary assume that there exist some vertex $x \in S_{i,l} \setminus S_i$. Then x is contained in C_i and C_l, and not in C_j. This is a contradiction since the subtree of T induced by the tree nodes that contain x is then not connected. ∎

The fact that $S_i = S_{i,l}$ ensures that, after the above described modification, T is still a legal clique tree of H. No maximal cliques are modified, and thus every vertex and every edge is still contained in some clique. Because $S_i = S_{i,l}$, every induced subtree split by deleting tree edge S_i is connected by the new edge $S_{i,l}$. It is also necessary to compute the new *add* and *remove* lists for $S_{i,l}$. In the general case we can assume that a sequence of tree nodes and tree edges are removed from the path $P_{u,v}$, and we want to compute the new *add* and *remove* list, which will be inserted between some clique C_i and C_l in $P_{u,v}$. The new $add(C_i, C_l)$ and $remove(C_i, C_l)$ can be computed in the following way:

- $add(C_i, C_l) = \bigcup_{i \leq q < l} add(C_q, C_{q+1}) \setminus \bigcup_{i < q < l} remove(C_q, C_{q+1})$.
- $remove(C_i, C_l) = \bigcup_{i \leq q < l} remove(C_q, C_{q+1}) \setminus \bigcup_{i < q < l} add(C_{q-1}, C_q)$.

We have now explained how $P_{u,v}$ can be reduced such that every tree edge on $P_{u,v}$ is a distinct minimal u, v-separator. Thus, we are done with the part that is illustrated in Figure 2(b)-(d). It is important to understand that, by Theorems 2.1, 2.2, and 2.3, every minimal u, v-separator of H is represented by a tree edge in $P_{u,v}$. However, it remains to explain how to examine adjacent tree edges in such a way that the total time bound $O(n)$ is maintained. The idea is to do this in two scans. One from C_u to C_v, and one from C_v to C_u. The same operation is done for both directions, so we will only explain the scan from C_u to C_v. Consider the tree edges in the order given by $P_{u,v}$. For a given tree edge S_i (starting from S_1) we will try to find the largest number t such that the intersection $S_{i,i+t} = C_i \cap C_{i+t}$ is equal to S_i. Note that S_i is then a subset of or equal to every edge between C_i and C_{i+t}. Since t is the largest such number, $S_{i,i+t} \not\subseteq S_{i+t,i+t+1} = S_{i+t}$. Thus, when we reach a tree edge S_{i+t} on $P_{u,v}$ such that $S_{i,i+t} \not\subseteq S_{i+t}$, we have found the largest such t for S_i. After that, we will remove the edge S_i between C_i and C_{i+1} from T, and add a new edge $S_{i,i+t}$ between C_i and C_{i+t}, which will shorten the path $P_{u,v}$ by removing possibly several tree nodes and edges from this path, so that the updated path will become $C_i, S_{i,i+t}, C_{i+t}, S_{i+t}, C_{i+t+1}, ...$ Then, we will continue with S_{i+t} as the new S_i.

The time bound is achieved using the following technique. To start with $t = 1$, we will decide if $S_{i,i+t} \subseteq S_{i+t}$, which is, due to Claim 5.3, equivalent to deciding if $remove(C_{i+t}, C_{i+t+1}) \subseteq add(C_{i+t-1}, C_{i+t})$ for $t = 1$. Let A be a vector of size n, where every element is set to 0. Read $add(C_{i+t-1}, C_{i+t})$ and set $A[x]$ to 1 for every vertex $x \in add(C_{i+t-1}, C_{i+t})$. Then for every vertex x in $remove(C_{i+t}, C_{i+t+1})$ check the value of $A[x]$. In the case where $A[x] = 1$ for every vertex $x \in remove(C_{i+t}, C_{i+t+1})$ we know that $remove(C_{i+t}, C_{i+t+1}) \subseteq \bigcup_{1 \leq q \leq t} add(C_{i+q-1}, C_{i+q}) \setminus \bigcup_{1 < q \leq t} remove(C_{i+q}, C_{i+q+1}) = add(C_i, C_{i+t})$. We can now increment the value of t and go back to the point, where $A[x]$ is set to 1 for every vertex $x \in add(C_{i+t-1}, C_{i+t})$. In the case where one or more $A[x]$ is 0, and $x \in remove(C_{i+t}, C_{i+t+1})$, then $remove(C_{i+t}, C_{i+t+1}) \not\subseteq \bigcup_{1 \leq q \leq t} add(C_{i+q-1}, C_{i+q}) \setminus \bigcup_{1 < q \leq t} remove(C_{i+q}, C_{i+q+1}) = add(C_i, C_{i+t})$. Then $C_i \cap C_{i+t+1} \subset S_{i,i+t+1}$ and we have found the largest value of t. Resetting the A vector is done by setting $A[x]$ to 0 for every vertex $x \in \bigcup_{1 \leq q \leq t} add(C_{i+q-1}, C_{i+q})$. If $t > 1$ then we will insert the tree edge $S_{i,i+t}$ and delete S_i. Use S_{i+t} as S_i and start over again, until C_v is reached.

The *add* lists in $P_{u,v}$ are read once when updating the A vector, and once when resetting the same vector. Computing $add(i, i+t)$ and $remove(i, i+t)$ requires reading the *add* lists. Thus, every *add* list is read four times. The *remove* lists are read once when they are checked against the A vector, and once for each of the new lists $add(i, i+t)$ and $remove(i, i+t)$. Thus, every *remove* list is read three times. Remember that the scan is done in both directions, and that the *add* lists are *remove* lists in the opposite direction. Thus, every list is read seven times. We have thus reached the goal of modifying T in $O(n)$ time so that every tree edge in $P_{u,v}$ is a distinct minimal u, v-separator.

Now we will see how to compute X. A pair ux belongs to $R(H,u,v)$ if there exists a minimal u,v-separator containing x. Our goal is to compute the set X of vertices, where $x \in X$ if $ux \in R(H,u,v)$. Observe that $C_1 \subseteq N(u)$, and thus, only the vertices not in S_1 are of interest. The path $P_{u,v}$ is already modified such that every tree edge is a minimal u,v-separator, and every minimal u,v-separator is a tree edge in this path. We can compute X in the following way: Start in C_u with an empty vertex set X. Then for $1 \leq i < k-1$ add the vertices contained in $add(C_i, C_{i+1}) \setminus remove(C_{i+1}, C_{i+2})$ to X. There might be vertices that are only contained in a single maximal clique C_i, and thus not contained in any tree edge. These vertices will be contained in both $add(C_i, C_{i+1})$ and $remove(C_{i+1}, C_{i+2})$. Thus, we obtain the desired set X in $O(n)$ time using the previously described scanning technique and the same type of characteristic vector A.

5.2 Modifying T to reflect the addition of uv and $R(H,u,v)$ to H

We now discuss how to update T if we decide to add uv and the set of required edges to H. Let H' denote the graph that results from adding uv and $R(H,u,v)$ to H. We will modify T to obtain a clique tree T' of H'. If u and v is contained in different components of H and T, then we update T in the following way. Find a clique K_v of T containing v, create a new clique K_{uv} containing the vertices $\{u,v\}$, and insert the tree edge K_vK_{uv}. If the vertex u was contained in some clique of T before the clique K_{uv} was created, then let K_u be any such clique, and insert the tree edge $K_{uv}K_u$. The add and remove lists for K_vK_{uv} and $K_{uv}K_u$ can be computed strait forward in $O(n)$ time. Let us assume that u and v are contained in the same connected component of T and H for the rest of this subsection.

In order to update T to reflect that u has now become a neighbor of v and of every vertex in X, we simply place u in every tree edge and every tree node appearing on $P_{u,v}$ in T. This is illustrated in Figure 2(e). However, we must check the resulting tree T' after doing so, because there might be a tree node C on this path containing a vertex q not appearing in any minimal u,v-separator, and in this case u was not supposed to be a neighbor of q. Detecting such a tree node C is easy because then q cannot appear in any other tree node of the path. For any such C, we remove u from C, and we introduce a new tree node C' that contains u and every vertex of C except the vertices that do not appear in any other tree node. Edges incident to C on $P_{u,v}$ are redirected to be incident to C' instead, and tree edge $C'C$ is added to give a clique tree T' that reflects the neighborhood relations of H' correctly. This is illustrated in Figure 2(f), where C_4 corresponds to the mentioned C. If C_u has become a subset of another maximal clique because of this operation, then we must correct T' accordingly. This is shown in Figure 2(g).

Let us now discuss the practical implementation of this in $O(n)$ time. First remove the vertex u from the $remove(C_1, C_2)$ list. This ensures that u belongs to every maximal clique on $P_{u,v}$. Let us now consider each maximal clique C_i, $2 \leq i \leq k$, in the order given by $P_{u,v}$. The first step is to decide if C_i contains any vertex q as described above. We know that no such vertex q appears in a tree edge of $P_{u,v}$, and that X is the union of the tree edges in $P_{u,v}$. Thus, $Q = add(C_{i-1,C_i}) \setminus (X \cup \{v\})$ is exactly the set of such vertices q that are only contained in C_i. If $Q = \emptyset$, then we add u to C_i. This is done by adding u to every $add(C_l, C_i)$ list, where $l \notin \{i-1, i+1\}$ and C_l is a neighbor of C_i outside of $P_{u,v}$. The value of i can now be incremented, such that the process can continue from the next tree node. In the case where $Q \neq \emptyset$, we have to create a new tree node $C'_i = C_i \cap (X \cup \{v\}) \cup \{u\}$, and a new tree edge $S_{i',i}$ between C'_i and C_i. This is done by simply creating the new lists $add(C_i, C'_i)$ and $remove(C_i, C'_i)$ as follows: $add(C_i, C'_i) = \{u\}$, since $C'_i \setminus C_i = \{u\}$, and $remove(C_i, C'_i) = Q$. The lists $add(C_{i-1}, C'_i)$, $remove(C_{i-1}, C'_i)$, $add(C'_i, C_{i+1})$ and $remove(C'_i, C_{i+1})$ are not created, but obtained by altering $add(C_{i-1}, C_i)$, $remove(C_{i-1}, C_i)$, $add(C_i, C_{i+1})$ and $remove(C_i, C_{i+1})$. This is done by moving the pointers from C_i to C'_i, and removing all the vertices in Q from the lists.

Let us show that the $O(n)$ time bound is kept during the modifications explained above. Creating each new maximal clique C'_i is a constant time operation. Every time a new C'_i is created, we also create a new tree edge $S_{i,i'}$. We first argue that the sum of the sizes of all the $add(C_i, C'_i)$ and $remove(C_i, C'_i)$ lists for all such new tree edges is $O(n)$. For each new C'_i, no vertex $q \in C_i \setminus (X \cup \{v\})$ is contained in any other tree node on $P_{u,v}$, since x would then be contained in some minimal u, v-separator and thus belong to X. The number of all such vertices q over all C_i is less than n, and thus the total cost of creating all such new tree edges $S_{i,i'}$ is $O(n)$. In order to move the tree edges $S_{i-1,i}$ and $S_{i,i+1}$ to $S_{i-1,i'}$ and $S_{i',i+1}$ we must change some pointers, and read through the lists to remove vertices in $C_i \setminus C'_i = Q$. The total cost of all such operations is less or equal to the sum of all *add* and *remove* lists in $P_{u,v}$. It follows that this altogether is an $O(n)$ time operation.

We will now, through the next three claims, prove that tree T' that results from the modifications explained above is a clique tree of $H' = (U \cup \{u\}, D \cup \{uv\} \cup R(H, u, v))$.

Claim 5.5 *Given a chordal graph $H = (V, D)$, a clique tree T of H, an edge uv, and the required set of edges $R(H, u, v)$. Let H' be the graph $(V, D \cup \{uv\} \cup R(H, u, v))$ and let T' be the resulting clique tree after updating T such that it represents H'. Then for each pair of vertices x and y, there is a tree node in T' that contains both x and y if and only if $xy \in D \cup uv \cup R(H, u, v)$.*

Proof. Before any modifications to T at this step, there is a tree node $C \in T$, that contains both the vertices x and y if and only if $xy \in D$. A tree node C_d is only deleted during the modification process if there exists a remaining tree node C'_d, such that $C_d \subseteq C'_d$. Thus, for every edge $xy \in D$ there exists a tree node in T' that contains both x and y.

Before appropriate tree nodes of T are expanded to contain u, every newly created tree node C'_i is a subset of some other tree node C_i. At this point we have the property that the vertex set of every tree node of T is either a maximal clique in H or a subset of a maximal clique in H. Thus T has still the property that there exists a tree node containing x and y if and only if $xy \in D$.

Then u is added to every maximal clique C of T on the modified path $P_{u,v}$, where $C \subseteq X \cup \{v\}$, and we obtain T' . It follows that for every edge $xy \notin E(H')$, there do not exist any tree node C of T' that contains both x and y, since u is only added to a tree node C if $C \subseteq X \cup \{v\}$.

For the other direction we have to show that for every edge $ux \in R(H, u, v) \cup \{uv\}$ there exists a tree node C in T' containing u and x. By Theorems 2.1, 2.2, and 2.3 every minimal u, v-separator is an edge of $P_{u,v}$, thus there exists a tree node C in T on $P_{u,v}$ containing x for every $ux \in R(H, u, v)$. The tree node C_v in the end of $P_{u,v}$ contains v. If $C \not\subseteq X \cup \{v\}$ for a tree node C in T on $P_{u,v}$, then a new tree node $C' = C \cap (X \cup \{v\})$ is created and used in the path $P_{u,v}$. We can now conclude that for every edge $ux \in R(H, u, v) \cup \{uv\}$ there exists a tree node of T' which contains both u and x. ∎

Claim 5.6 *Subtree T'_x induced by the tree nodes in T' that contain vertex x is connected, for every vertex $x \in U \cup \{u\}$.*

Proof. We assume that all subtrees are connected in T before the last modification. Let us now consider the operations one by one. First operation is when $C_i \notin (X \cup \{v\})$. A new maximal clique C'_i is created, where $C'_i \subset C_i$. A tree edge is inserted between C_i and C'_i, but all subtrees are connected since $C'_i \subset C_i$. Next step is to move the edges $S_{i-1,i}$ and $S_{i,i+1}$ to become $S_{i-1,i'}$ and $S_{i',i+1}$. This will not create separated subtrees since $S_{i-1,i} \cup S_{i,i+1} \subseteq C'_i$, thus $S_{i-1,i'} = S_{i-1,i}$ and $S_{i',i+1} = S_{i,i+1}$. The second operation is adding the vertex u to C'_i in the case where a new tree node C'_i is created, and to C_i if no new tree node is created. This changes only the tree induced by the tree nodes containing the vertex u. Since we consider the maximal cliques in the order C_2 to C_v, then it follows that the tree T_u is always connected. ∎

Claim 5.7 *Every tree node except C_u in T' is a unique maximal clique in H'.*

Proof. There are two cases. The first is when the vertex u is added to a maximal clique C_i. The expanded C_i cannot become a subset of another tree node, but it can become a superset of a tree node C_j, if $C_j \setminus C_i = \{u\}$. Since C_u is the only tree node in the neighborhood of any tree node in $P_{u,v}$ that contains u, then this can only happen to C_u. The second case is when a new maximal clique C'_i is created. C'_i is at first a subset of C_i, but this is not a problem since u is later added to C'_i and not to C_i. C'_i is inserted between C_{i-1} and C_{i+1} in $P_{u,v}$, and u is added to all three cliques. Let us on the contrary assume that C'_i is a subset of one of its neighbors. Without loss of generality we can assume that $C'_i \subseteq C_{i+1}$. Then $remove(C'_i, C_{i+1}) = \emptyset$, and thus $S_{i-1,i'} \subseteq S_{i',i+1}$ since $remove(C'_i, C_{i+1}) \subseteq add(C_{i-1}, C'_i)$, which is a contradiction since every tree edge in $P_{u,v}$ is a unique minimal u, v-separator. ∎

Let us re-sort the maximal cliques of the modified path $P_{u,v}$ of T' from $C_u = C_1$ to $C_k = C_v$. With the above three claims, if $C_u \not\subseteq C_2$, then we have proved that T' is a proper clique tree of H'. If $C_u \subseteq C_2$, then we will simply remove C_u, and again we can conclude that T' with this final modification is a proper clique tree of H'.

However, it remains to explain how this final update of removing $C_u = C_1$ can be done in $O(n)$ time, which is challenging. It is easy to check if $C_1 \subseteq C_2$, since $remove(C_1, C_2) = \emptyset$ in this case. The clique C_1 is deleted in the following way: For every tree edge $S_{1,j}$ where $C_j \neq C_2$, we delete the tree edge $S_{1,j}$ and insert $S_{2,j}$. Afterward we delete the tree edge $S_{1,2}$ and the clique C_1. In order to do this efficiently we actually alter the *add* and *remove* lists and move the tree edges from C_1 to C_2. Let us consider the clique C_j, and how to create the *add* and *remove* lists from C_j to C_2. From the previous described technique they can be computed as follows: $add(C_j, C_2) = add(C_j, C_1) \cup add(C_1, C_2) \setminus remove(C_1, C_2)$ and $remove(C_j, C_2) = remove(C_1, C_2) \cup remove(C_j, C_1) \setminus add(C_j, C_1)$. Remember that $remove(C_1, C_2) = \emptyset$, since $C_1 \subseteq C_2$, and that $remove(C_j, C_1) \cap add(C_j, C_1) = \emptyset$. Computing the lists can then be reduced to: $add(C_j, C_2) = add(C_j, C_1) \cup add(C_1, C_2)$ and $remove(C_j, C_2) = remove(C_j, C_1)$. The obstacle regarding the time complexity is that $add(C_1, C_2)$ will be read once for each neighbor of C_1. Thus, we have to ensure that this work does not sum up to more than $O(n)$. Let us count the number of times this can happen.

Claim 5.8 *Let $C_u \subseteq C_2$ in T', and let T'' be the clique tree right before C_u occurred. Then $C_u \setminus \{u\}$ is not a clique in T''.*

Proof. The clique C_u was created when updating T'' with some edge uv' and the set $R(H'', u, v')$. The clique C_2 existed in T'' since only edges incident to u are handled since then. Let us on the contrary assume that $C_u \subseteq C_2$ Now $C_u \setminus \{u\}$ can not be in T'', else $C_u \setminus \{u\}$ is a subset of C_2 in T'', which is a contradiction.

Let us now on the contrary assume that u was added to some C and not a new clique C' to obtain the clique C_u. From using the same argument as for C_2, we know that C exist in T''. This is a contradiction since $C \subseteq C_2$. ∎

Claim 5.9 *Reducing the path $P_{u,v}$, such that it contains only distinct minimal u, v-separators, can increase the degree of C_u by at most 1.*

Proof. Two different scans are done on $P_{u,v}$ to reduce the number of cliques. The first starts in C_u, and finds the maximal clique furthest from C_u that is a superset of $S_{u,2}$. In this case one tree edge incident to C_u is deleted, and one is created, and the degree of C_u remains the same. In the direction from C_v to C_u, we may find a tree edge that is a subset of $S_{u,2}$. In this case C_u gets a new neighbor and the degree of C_u increases by 1. ∎

Observe that the process of reducing the path $P_{u,v}$ can increase the degree of at most one tree node containing the vertex u. This follows from the fact that C_u is the only tree node in $P_{u,v}$ containing the vertex u.

Claim 5.10 *Adding u to every tree node in $P_{u,v}$ does not increase the degree of C_u.*

Proof. One of two things will happen. In one case vertex u is added to C_2, which is the neighbor of C_u in the path $P_{u,v}$. This will not change the degree of C_u in the clique tree T. The second case is if some subset of the vertices in C_2 is not contained in the set X. Then a new clique C_2' is created, and the tree edge between C_u and C_2 is removed, and inserted between C_u and C_2. It follows that the degree of C_u is unchanged. ∎

Claim 5.11 *The degree of each newly created tree node C_i' in T' is at most* 3.

Proof. When a new clique C_i' is created, it is a subset of an existing clique C_i. Let k be the number of neighbors C_i has in $P_{u,v}$. Thus, k is either 1 or 2. A tree edge is introduced between C_i' and C_i, and C_i' replaces C_i in the path $P_{u,v}$. The degree of C_i' becomes $k+1$, and thus the degree is at most 3. ∎

Remember that the obstacle in obtaining the $O(n)$ time bound was that $add(C_u, C_2)$ is read once for each neighbor of C_u, when $add(C_j, C_2) = add(C_j, C_u) \cup add(C_u, C_2)$ is computed. Observe that $add(C_j, C_u) \cap add(C_u, C_2) = \emptyset$. This follows from the way the data structure is defined. So we can compute $add(C_j, C_2)$ by adding $add(C_u, C_2)$ to $add(C_j, C_u)$, without even reading the $add(C_j, C_u)$ list. Let us now consider a single $add(C_j, C_u)$, and assume that $add(C_u, C_2)$ is added to obtain $add(C_j, C_2)$. Let us further assume that C_2 becomes a subset of some C_2'' at some later step. Then some list $add(C_2, C_2'')$ will be added to $add(C_j, C_2)$, and this can continue for many iterations. It is important now to remember that $add(C_j, C_u) \cap add(C_u, C_2) = \emptyset$, and thus only $O(n)$ vertices can be added to each such add list. We conclude this with an amortized time analysis. The idea is to let each introduced edge uv be responsible for some tree edges. The number of tree edges an edge is responsible for can be argued in the following way: The operation of reducing the path $P_{u,v}$ makes uv responsible for one edge, due to Claim 5.9. Let us assume that $C_u \subset C_2$, when the edges uv and $R(H, u, v)$ are inserted. Then the edge uv becomes responsible for the edges incident to C_u, right after it was created. Claim 5.11 ensures that this number is at most 3. The final operation that can effect the degree of C_u is if C_u was C_2' for some C_u', which was deleted. In this case C_u inherits the neighbors of C_u'. This is not a problem since we can recursively use these tree arguments on C_u'. To conclude, each edge uv is responsible for at most 4 edges, and at most n vertices can be added to the *add* lists in these tree edges. It follows that deleting C_u can be done in amortized $O(n)$ time for each vertex uv.

6 Concluding remarks

In this paper, we contribute new theoretical results on chordality as well as an efficient handling of the corresponding data structures. Not only do we have a new $O(nm)$ time on-line algorithm for minimal triangulation of a graph G, but we are able to compute at the same time a maximal chordal subgraph, thus "minimally sandwiching" the graph between two chordal graphs: $H_1 \subseteq G \subseteq H_2$.

This special feature of our algorithm enables the user, at no extra cost, to choose at each vertex addition step whether he wants to *add* or *delete* edges, or even to do so at each edge addition step. This may be interesting for applications such as updating databases or for sampling techniques in the context of artificial intelligence when maintaining a chordal graph is required or desirable.

Recent work has shown that minimal separation plays an important role in the process of minimal triangulation. Our new characterization of chordal graphs, which uses minimal separation, leads us to believe that there is a corresponding relationship when computing a maximal chordal subgraph.

A continuation of this work would be to compare the running time of our algorithm to other minimal triangulation algorithms with experimental tests. Since often several edges are found and inserted at the time cost of one edge, we conjecture that our algorithm may be very fast in practice. Another important issue to inquire about would be how well our algorithm performs when used as a heuristic for hard problems, such as computing a minimum triangulation or a maximum subtriangulation. Standard ideas from existing heuristics, like picking a vertex of minimum degree at each step, could be integrated into our algorithm and possibly result in higher probability of less fill in minimal triangulations and more edges in maximal subtrinagulations. !!!!!!!!!!!!!! WHAT ARE WE TRYING TO SAY HER ??? !!!!!!!!!!!!!!!!!!!!!!!!!!! We suspect, in fact, that a *maximum* admissible set of the original edges between the added vertex and the current chordal subgraph can be computed at each step at additional polynomial cost. !!!

It appears that chordal graphs are in many ways similar to weakly chordal graphs [22, 7, 6]. It would be interesting to extend our results to define a process which maintains a weakly chordal graph, thus enabling efficient computation of a weak minimal super or sub triangulation, which is an important issue for recent applications to formal concept analysis and data mining [9]. As we pointed out in Section 4, the required set of edges can be seen as a succession of 2-pairs which is computed efficiently. In view of the important role that 2-pairs play in weakly chordal graph recognition [23, 32, 24], our results could possibly be extended to efficiently handle such a succession of 2-pairs in a weakly chordal graph, with the hope of improving the current $O(m^2)$ [24] time complexity for this problem.

References

[1] S. Arikati and P. Rangan. An efficient algorithm for finding a two-pair, and its applications. *Disc. Appl. Math.*, 31:71–74, 1991.

[2] E. Balas. A fast algorithm for finding an edge-maximal subgraph with a TR-formative coloring. *Disc. Appl. Math.*, 15:123–134, 1986.

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

[4] A. Berry. A wide-range efficient algorithm for minimal triangulation. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[5] A. Berry, J. Blair, P. Heggernes, and B Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39(4):287–298, 2004.

[6] A. Berry and J-P. Bordat. Triangulated and weakly triangulated graphs: Simpliciality in vertices and edges. *6th International Conference on Graph Theory (ICGT 2000).* Communication.

[7] A. Berry, J-P. Bordat, and P. Heggernes. Recognizing weakly triangulated graphs by edge separability. *Nordic Journal of Computing*, 7:164–177, 2000.

[8] A. Berry, J-P. Bordat, P. Heggernes, G. Simonet, and Y. Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. Technical Report Reports in Informatics 243, University of Bergen, Norway, 2003. Submitted to *Journal of Algorithms*.

[9] A. Berry and A. Sigayret. Obtaining and maintaining polynomial-size concept lattices. In *Proceedings of FCAKDD, (ECAI 2002)*, pages 3–6, 2002.

[10] A. Berry, A. Sigayret, and C. Sinoquet. Maximal sub-triangulation as improving phylogenetic data. Technical Report RR-02-02, LIMOS, Clermont-Ferrand, France, 2002.

[11] J. R. S. Blair, P. Heggernes, and J. A. Telle. A practical algorithm for making filled graphs minimal. *Theoretical Computer Science*, 250:125–141, 2001.

[12] 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.

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

[14] T. F. Coleman. A chordal preconditioner for large-scale optimization. *Applied Math.*, 40:265–287, 1988.

[15] E. Dahlhaus. Minimal elimination ordering inside a given chordal graph. In R. H. Möhring, editor, *Graph Theoretical Concepts in Computer Science - WG '97, LNCS 1335*, pages 132–143. Springer Verlag, 1997.

[16] P. M. Dearing, D. R. Shier, and D. D. Warner. Maximal chordal subgraphs. *Disc. Appl. Math.*, 20:181–190, 1988.

[17] A. Deshpande, M. Garofalakis, and M. I. Jordan. Efficient stepwise selection in decomposable models. In *Proceedings of UAI 2001*, pages 128–135.

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

[19] P. Erdös and R. Laskar. On maximum chordal subgraph. *Cong. Numerantium*, 39:367–373, 1983.

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

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

[22] R. Hayward. Generating weakly triangulated graphs. *J. Graph Theory*, 21:67–70, 1996.

[23] R. Hayward, C. Hoàng, and F. Maffray. Optimizing weakly triangulated graphs. *Graphs and Combinatorics*, 5:339–349, 1989.

[24] R. Hayward, J. Spinrad, and R. Sritharan. Weakly chordal graph algorithms via handles. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[25] P. Heggernes and Y. Villanger. Efficient implementation of a minimal triangulation algorithm. In R. H. Möhring, editor, *Algorithms - ESA 2002, LNCS 2461*, pages 550–561. Springer Verlag, 2002.

[26] 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.

[27] L. Ibarra. Fully dynamic algorithms for chordal graphs. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[28] M. Lundquist. *Zero patterns chordal graphs and matrix completions.* PhD thesis, Clemson University, USA, 1990.

[29] A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Disc. Appl. Math.*, 113:109–128, 2001.

[30] 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.

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

[32] J. Spinrad and R. Sritharan. Algorithms for weakly triangulated graphs. *Disc. Appl. Math.*, 59:181–191, 1995.

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

[34] J. Xue. Edge-maximal triangulated subgraphs and heuristics for the maximum clique problem. *Networks*, 24:109–120, 1994.

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